Spring 2009 - Real-Time Systems

http://www.neu-rtes.org/courses/spring2009/

Chapter 7
Real-time Systems on Multi-Cores

Real-Time Embedded Systems Laboratory
Northeastern University

Let's begin with an ooooooold story

- once upon a time ……

Let's begin with an ooooooold story

- there was a monk ……

Let's begin with an ooooooold story

- then comes another monk ……

Let's begin with an ooooooold story

- then comes the third monk ……

Let's begin with an ooooooold story

- What do we learn?

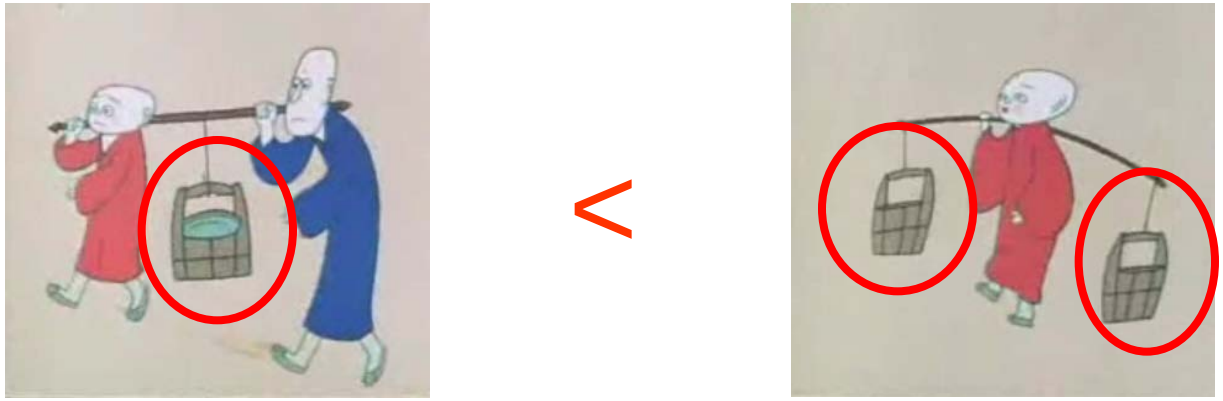
More workers may lead to less production, if they can not cooperate well!

Open Question:

- What happens if there are 400 monks?

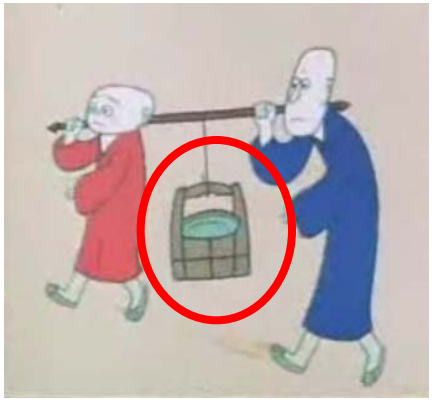
Welcome to the Cruel Multi-core World!

- Have you heard that *an application running on a dual-core chip can be actually slower than running on only one of them?*

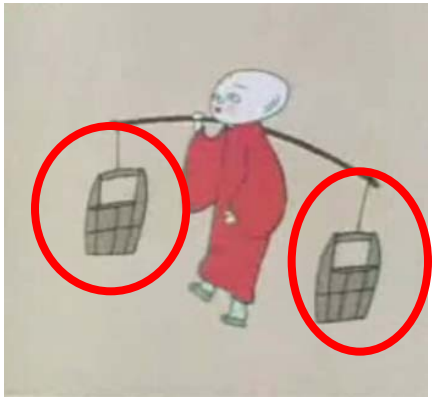


Welcome to the Cruel Multi-core World!

- Have you heard that *an application running on a dual-core chip can be actually **slower** than running on only one of the two cores?*



<



- Unfortunately, that's a real story.

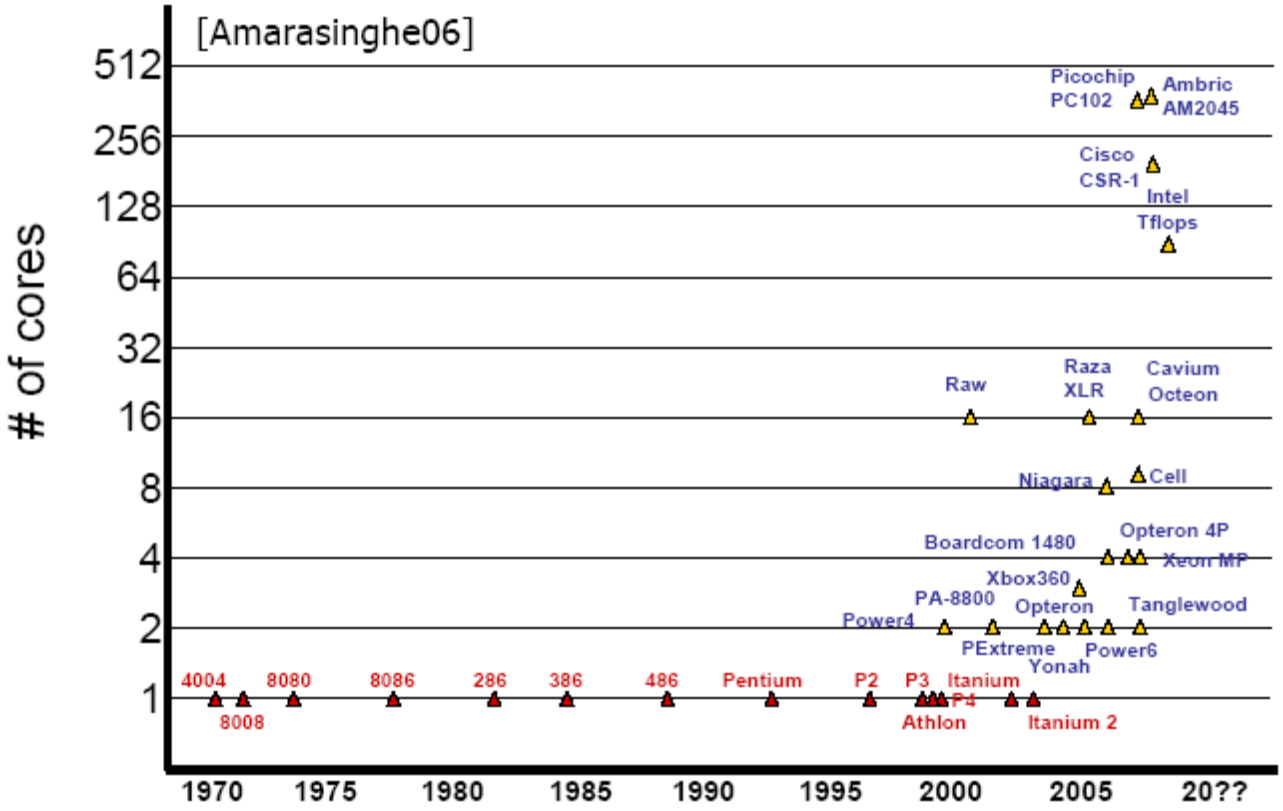
  according to a case-study of ABB robotic production application

Welcome to the Cruel Multi-core World!

- Today, PicoChip is selling chips with more than 400 DSP cores on it.
- Anant Agarwal (CEO of Tilera, also a professor in MIT) predicates there will be more than 4000 cores on embedded processor chips by 2017.
- ……

Multi-core is Booming

- for higher performance, lower power consumption, lower cost, smaller size … …



Everyone is doing it!

- *"Intel have 10 projects in the works that contain four or more computing cores per chip"*

  [Paul Otellini, Intel Chief Executive at IDF fall 2005]

- *"Today, processors with multiple CPUs and a large cache on a single chip are becoming common.*
- *Attempts to tease the parallelism out of a sequential program automatically haven't worked out very well.*
- *We need better education, better languages, and better tools, since building concurrent programs is hard"*
- [Andrew Herbert, Director of Microsoft Cambridge Research Lab, May 2005]

This Chapter

- Theoretic Aspect:

  Multiprocessor scheduling foundations

  (boring, but important to get insights!)

- Practical Aspect:

  Interesting issues of building real time systems on multi-core processors

This Chapter

- Theoretic Aspect:
  Multiprocessor scheduling foundations
  (boring, but important to get insights!)

- Practical Aspect:
  Interesting issues of building real time systems on multi-core processors

Task Model

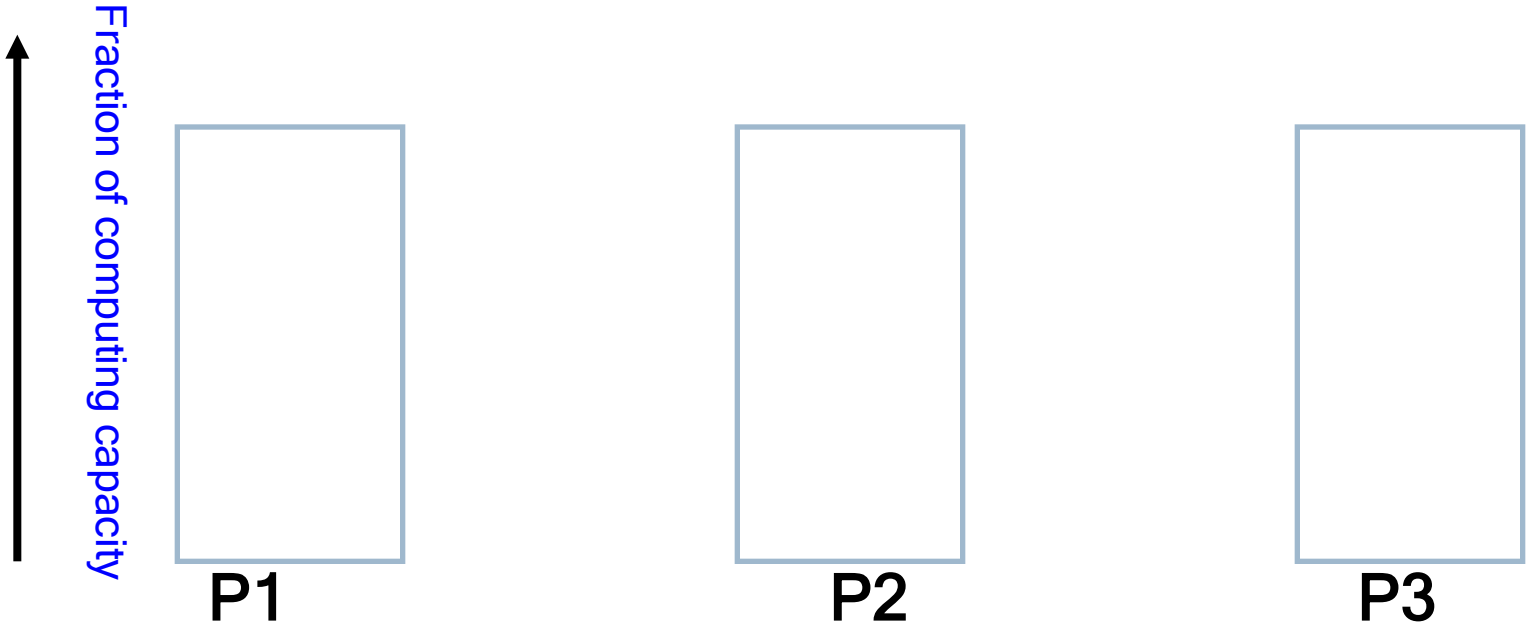
- Independent periodic task set: n tasks
- For each task t_i:
  - C_i: execution time
  - D_i: relative deadline
  - T_i: period
- Each task consists of a (infinite) sequence of jobs

  For each job j_i^k (the k^{th} job of task t_i):
  - r_i: release time
  - d_i: relative deadline

Multiprocessor Model

- *m* processors
- Identical multiprocessors:
  - each processor has the same computing capacity
- Uniform multiprocessors:
  - different processors have different computing capacities
- Heterogeneous multiprocessors:
  - each (task, processor) pair may have a different computing capacity

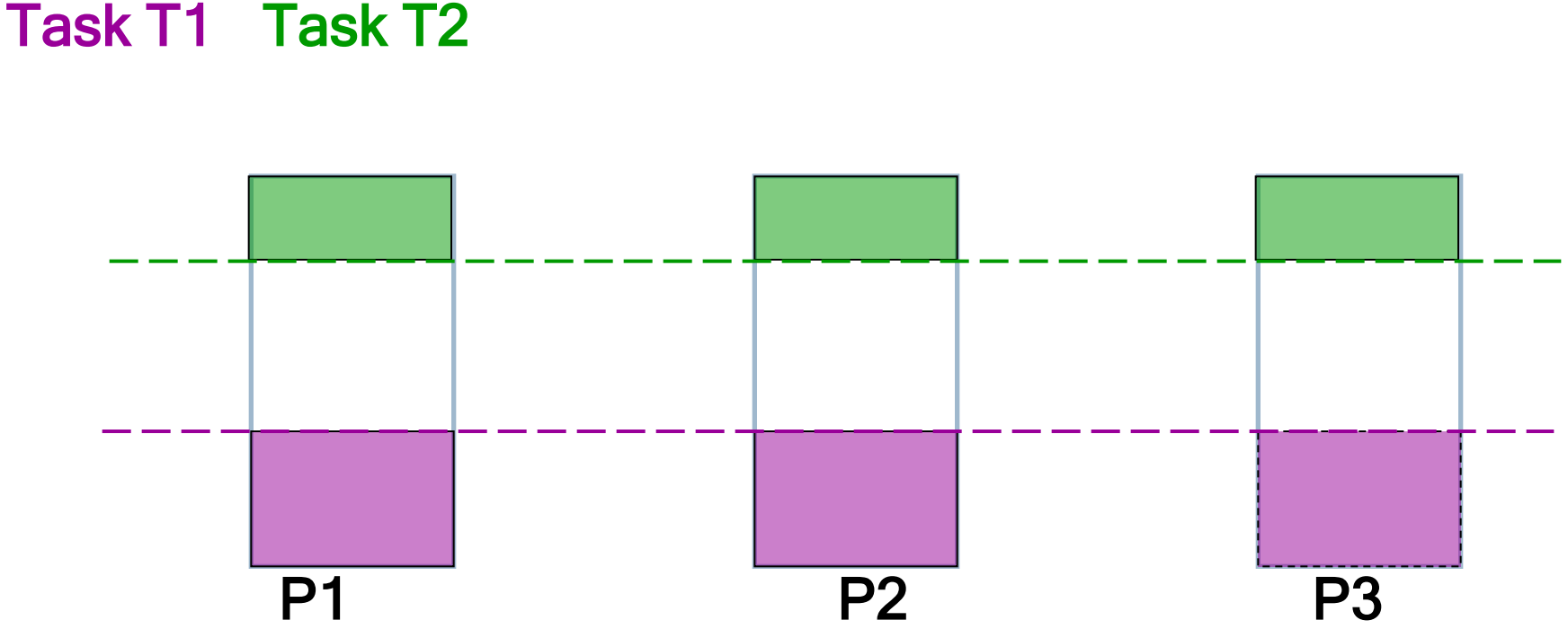
Multiprocessor Model



Multiprocessor Model

- Identical multiprocessors:

  each processor has the same computing capacity

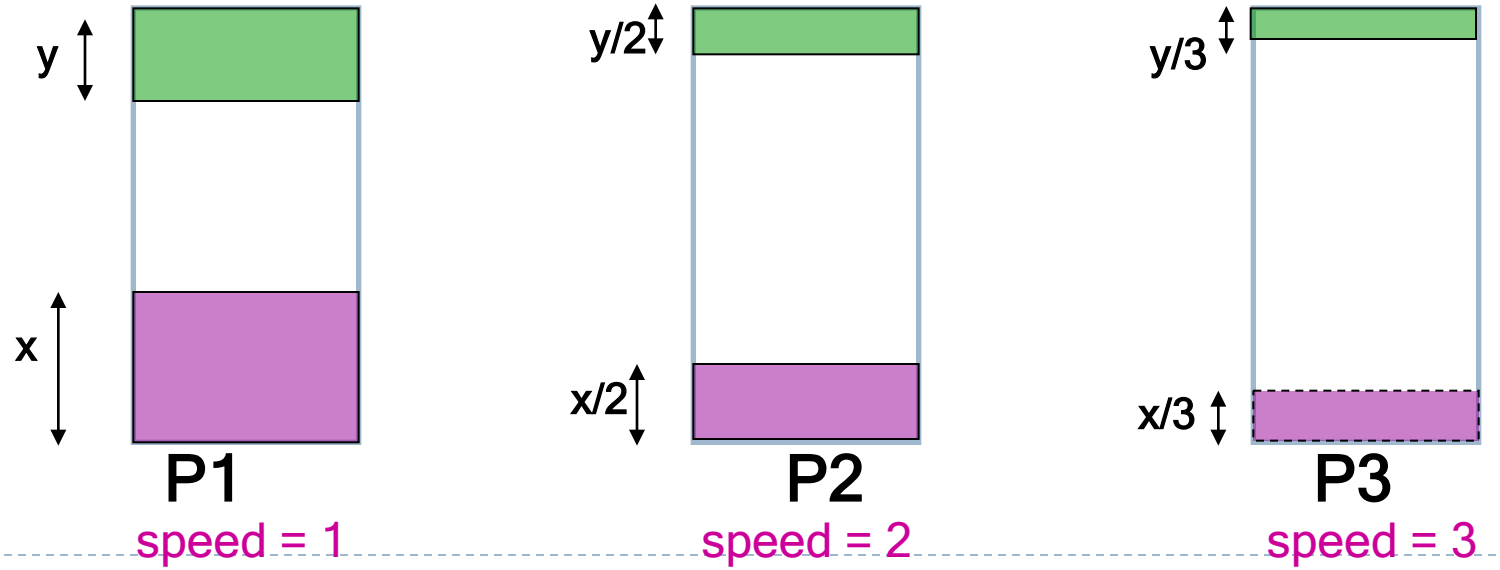


Multiprocessor Model

- Uniform multiprocessors:

  different processors have different computing capacities

Task T1 Task T2

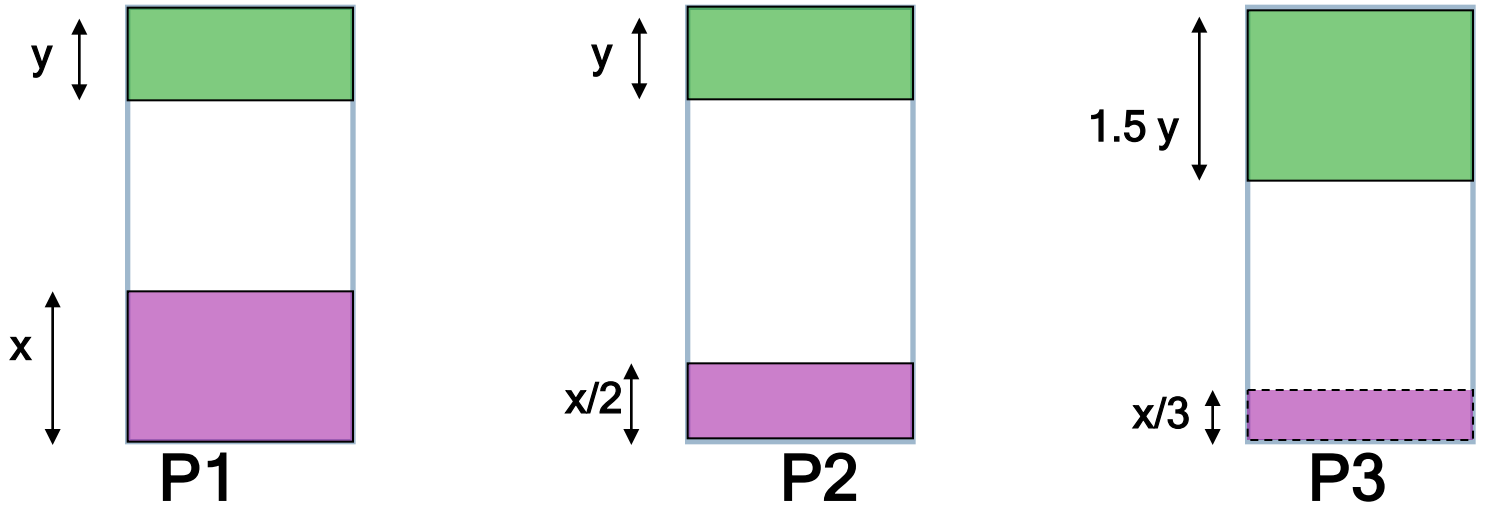


Multiprocessor Model

- Heterogeneous multiprocessors:

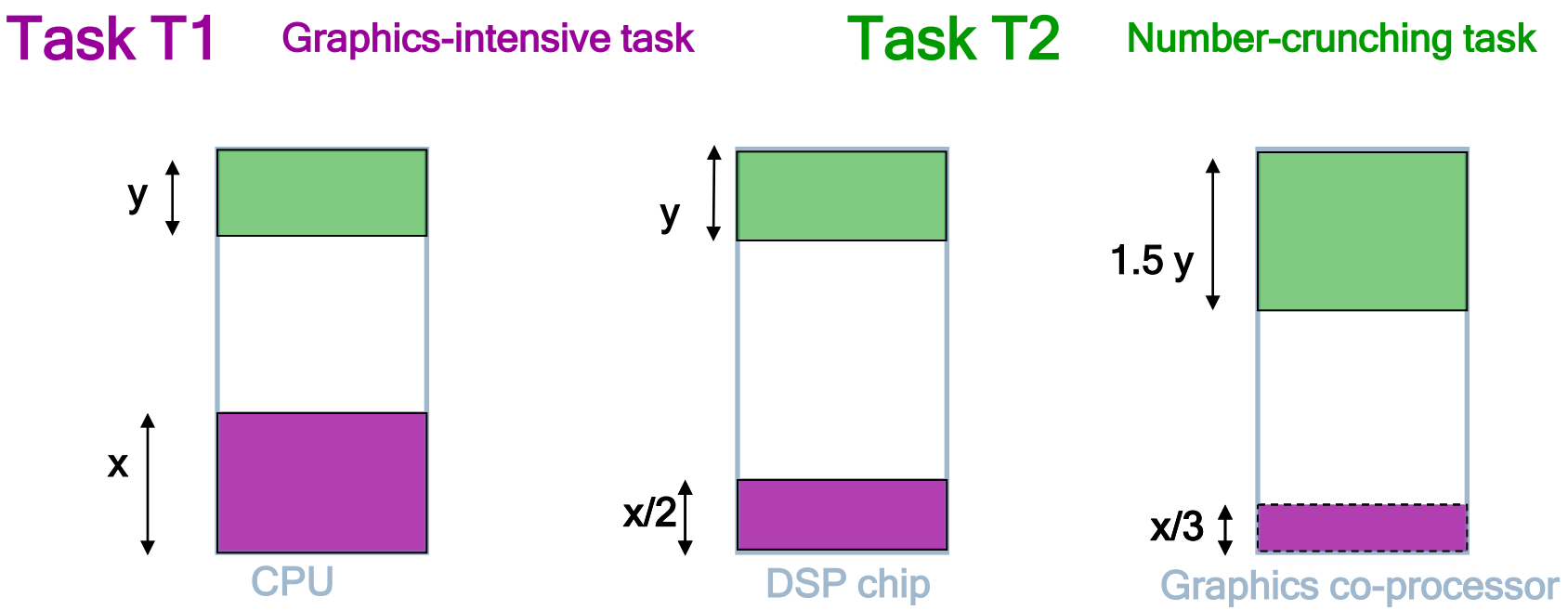
  each (task, processor) pair may have a different computing capacity

Task T1 Task T2



Multiprocessor Model

- Why study heterogeneous multiprocessors:
  - systems synthesized using specialized COTS processors

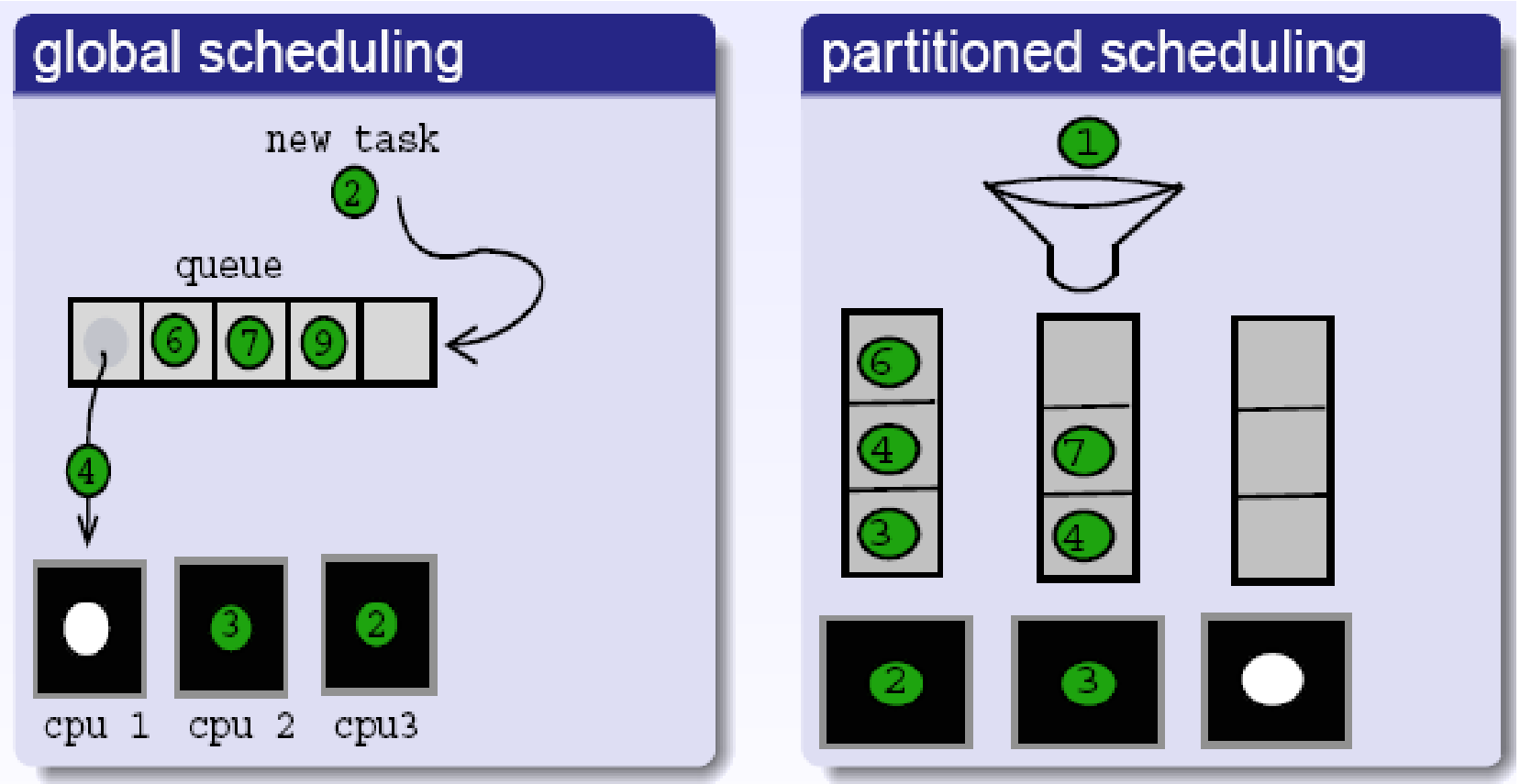


Classification of Multiprocessor Scheduling

- According to migration:
  - Partitioned scheduling (no-migration)
    - Each task may only execute on a specific processor
  - Global scheduling (full-migration)
    - Any task's job may execute on any processor
  - Middle approach (restrict-migration)
    - Each job is assigned to a single processor, while a task is allowed to migrate.
    - In other words, inter-processor task migration is permitted only at job boundaries.

Classification

- According to migration:



Classification

- According to priority assignment
  - Static priorities
    - A unique priority is associated with each task, and all jobs generated by a task have the priority associated with that task
    - E.g. RM
  - Job-level dynamic priorities:
    - For every pair of jobs J_i and J_j , if J_i has higher priority than J_j at some instant in time, then J_i always has higher priority than J_j .
    - EDF: proven optimal for uniprocessor scheduling
  - Unrestricted dynamic priorities
    - Relative priority of two jobs may change at any time.
    - LLF: more optimal than EDF for MP scheduling

Classification

	1: static	2: job-level static	3: full dynamic
1: partitioned	(1, 1)	(1, 2)	(1,3)
2: restrict migration	(2,1)	(2,2)	(2,3)
3: full migration	(3,1)	(2,3)	(3,3)

Classification

- Work-conserving scheduling
  - a processor is never left idle while an active job exists
  - e.g. Global EDF, Global RM, . . .
- Non-conserving scheduling
  - a processor could be idle while an active job exists
  - e.g. Restrict-migrate EDF, Partitioned EDF, . . .

Partitioned Scheduling

- Advantages:
  - Mature scheduling framework
  - – Most scheduling theory pertaining to uniprocessor scheduling are also applicable here
  - – Uniprocessor resource-management protocols can be used
- Partitioning of tasks can be automated
  - – For example, using a bin-packing algorithm
- Disadvantages:
  - Cannot exploit all unused execution time
    - – Surplus capacity cannot be shared among processors

Partitioned Scheduling

- Complexity of schedulability analysis for partitioned scheduling:
  - The problem of deciding whether a task set is schedulable on *m* processors with respect to partitioned scheduling is NP-complete

- Consequence:
  - There cannot be any pseudo-polynomial time algorithm for finding an optimal partition of a set of tasks unless P = NP.

Partitioned Scheduling

- Bin-packing algorithms:
  - The problem concerns packing objects of varying sizes in boxes (”bins”) with the objective of minimizing number of used boxes.
- Application to multiprocessor systems:
  - Bins are represented by processors and objects by tasks.
  - The decision whether a processor is ”full” or not is derived from a utilization-based feasibility test.
- Assumptions:
  - Independent, periodic tasks
  - Preemptive, uniprocessor scheduling (RM)

Partitioned scheduling

- Bin-packing algorithms
- Rate-Monotonic-First-Fit (RMFF):
  - Let the processors be indexed as N1, N2, ...
  - Assign the tasks in the order of increasing periods (that is, RM order).
  - For each task *i*, choose the lowest previously-used processor *n* such that *i*, together with all tasks that have already been assigned to processor *n*, can be feasibly scheduled according to the utilization-based RM-feasibility test.

Global Scheduling

- General characteristics:
  - All ready tasks are kept in a common (global) queue
  - When selected for execution, a task can be dispatched to an arbitrary processor, even after being preempted
  - Task execution is assumed to be ”greedy”:
    - If higher-priority tasks occupy all processors, a lower-priority task cannot grab a processor until the execution of a higher priority task is complete.

Global Scheduling

- Advantages:
  - Supported by most multiprocessor operating systems
    - Windows NT, Solaris, Linux, ...
  - Effective utilization of processing resources
    - Unused processor time can easily be reclaimed
- Disadvantages:
  - Weak theoretical framework
    - Few results from the uniprocessor case can be used
  - Poor resource utilization for hard timing constraints
    - No more than 50% resource utilization can be guaranteed
  - Suffers from several scheduling anomalies
    - Sensitive to period adjustments

Global Scheduling

- The ”root of all evil” in global scheduling: (Liu, 1969)

> Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that *a task can use only one processor even when several processors are free at the same time* adds a surprising amount of difficulty to the scheduling of multiple processors

Global Scheduling

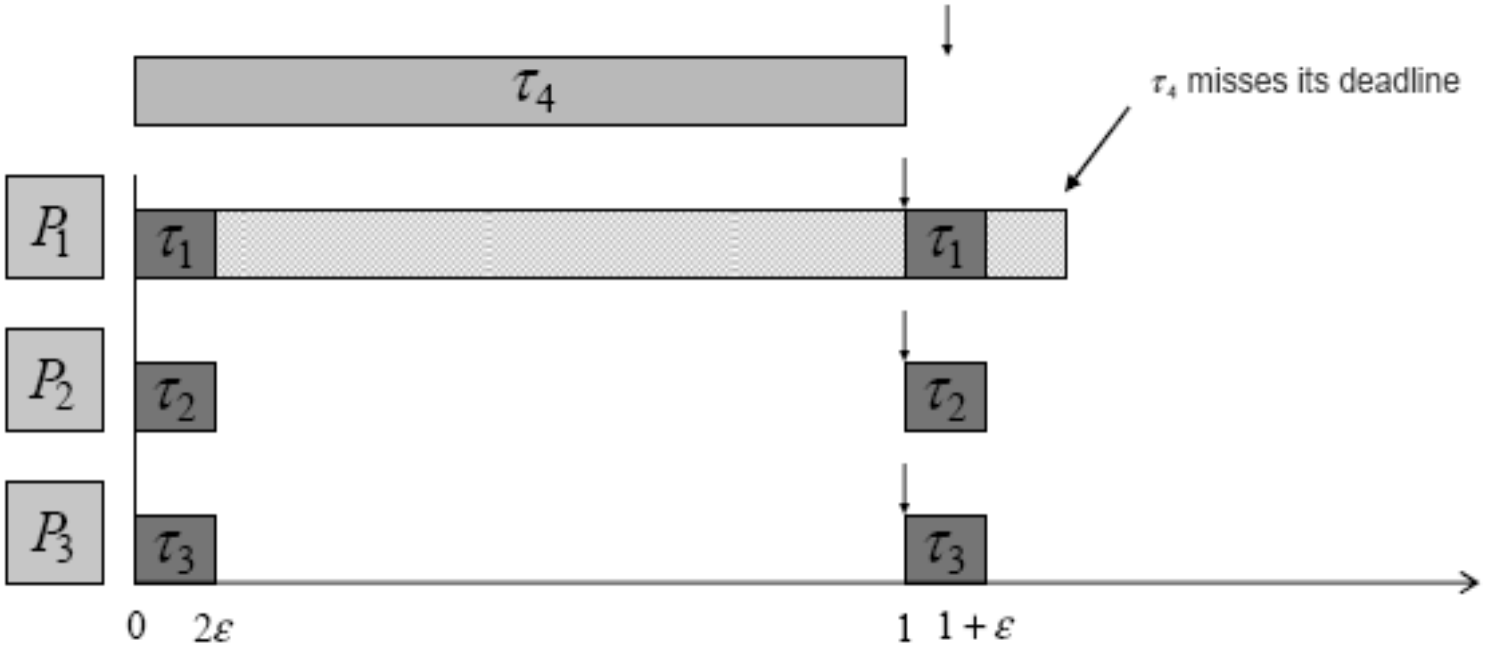
- Dhall's effect:
  - With work-conserving scheduling algorithms, some low-utilization task sets can be unschedulable regardless of how many processors are used.
- Dependence on relative priority ordering:
  - Changing the relative priority ordering among higher-priority tasks may affect schedulability for a lower-priority task.
- Hard-to-find critical instant:
  - A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks' release time.

Dhall's Effect

Dhall's effect: (Dhall & Liu, 1978)

$$\tau_1 = \{C_1 = 2\varepsilon, T_1 = 1\}$$
$$\tau_2 = \{C_2 = 2\varepsilon, T_2 = 1\}$$
$$\tau_3 = \{C_3 = 2\varepsilon, T_3 = 1\}$$
$$\tau_4 = \{C_4 = 1, T_4 = 1+\varepsilon\}$$

(RM scheduling)



Dhall's Effect

Dhall's effect:

- Applies for (greedy) RM, DM and EDF scheduling
- Least utilization of unschedulable task sets can be arbitrarily close to 1 no matter how many processors are used.

$$U_{global} = m\frac{2\varepsilon}{1} + \frac{1}{1+\varepsilon} \to 1$$

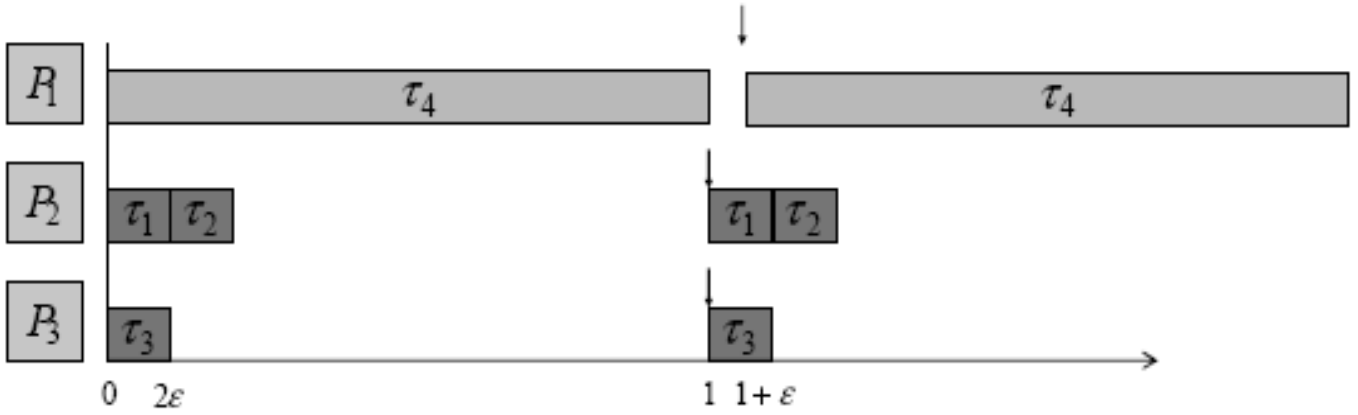
when $\varepsilon \to 0$

Consequence:

New multiprocessor priority-assignment schemes are needed!

Dhall's Effect

- Problem: RM, DM and EDF only account for task periods!
  - Actual computation demands are not accounted for.
- Solution: Dhall's effect can easily be avoided by letting tasks with high utilization receive higher priority:



Priority Assignment

- Impact of relative priority ordering:
  - The response time of a task depends on the relative priority ordering of the higher-priority tasks
  - This property does not exist for a uniprocessor system
  - This means that well-known uniprocessor methods for finding optimal priority assignments (for example RM, EDF) cannot be applied
- Consequence:
  - New methods for constructing optimal multiprocessor priority assignments are needed!

Priority Assignment

- Algorithm RM-US[m/(3m-2)]:
  - RM-US[m/(3m-2)] assigns (static) priorities to tasks according to the following rule:

    1) if $U_i > m/(3m-2)$ then task i has the highest priority (ties broken arbitrarily)

    2) if $U_i \leq m/(3m-2)$ then task i has RM priority

  - Clearly, tasks with higher utilization get higher priority

Priority Assignment

- RM-US[m/(3m-2)] example:
  - As an example of the priorities assigned by RM-US[m/(3m-2)], consider the following task set to be scheduled on a system with 3 identical processors:

    task 1 = $\{C_1=1, T_1=7\}$ $(U_1=0.143)$

    task 2 = $\{C_2=2, T_2=10\}$ $(U_2=0.2)$

    task 3 = $\{C_3=9, T_3=20\}$ $(U_3=0.45)$

    task 4 = $\{C_4=11, T_4=22\}$ $(U_4=0.5)$

    task 5 = $\{C_5=2, T_5=25\}$ $(U_5=0.08)$

Priority Assignment

- RM-US[m/(3m-2)] example:
  - For m-3

    $m/(3m-2) = 3/7 \approx 0.4286$

  - Hence, tasks 3 and 4 will be assigned higher priorities, and remaining tasks will be assigned RM priorities.

  - The possible priority assignments are therefore as follows (highest-priority task listed first):

    3, 4, 1, 2, 5 or 4, 3, 1, 2, 5

Priority Assignment

- Processor Utilization analysis for RM-US[m/3m-2]
  - A sufficient condition for RM-US[m/(3m-2)] scheduling on m identical processor is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \frac{m^2}{3m-2}$$

  - Question: does RM-US[m/(3m-2)] avoid Dhall's effecit?

Priority Assignment

- Processor Utilization analysis for RM-US[m/3m-2]

$$U_{RM-US[m/(3m-2)]} = \lim_{m \to \infty} \frac{m^2}{3m-2} = \frac{m}{3}$$

  - Regardless of the number of processors, the task set will always meet its deadlines as long as no more than one third of the processing capacity is used.
  - RM-US[m/(3m-2)] thus avoid Dhall's effect since we can always add more processors if deadlines were missed.
  - Note that this remedy was not possible with pure RM.

Response Time Analysis

- Response-time analysis for multiprocessors:
  - Uses the same principle as the uniprocessor case, where the response time for a task i consists of;

    C_i: The task's WCET

    I_i: Interference form higher-priority tasks

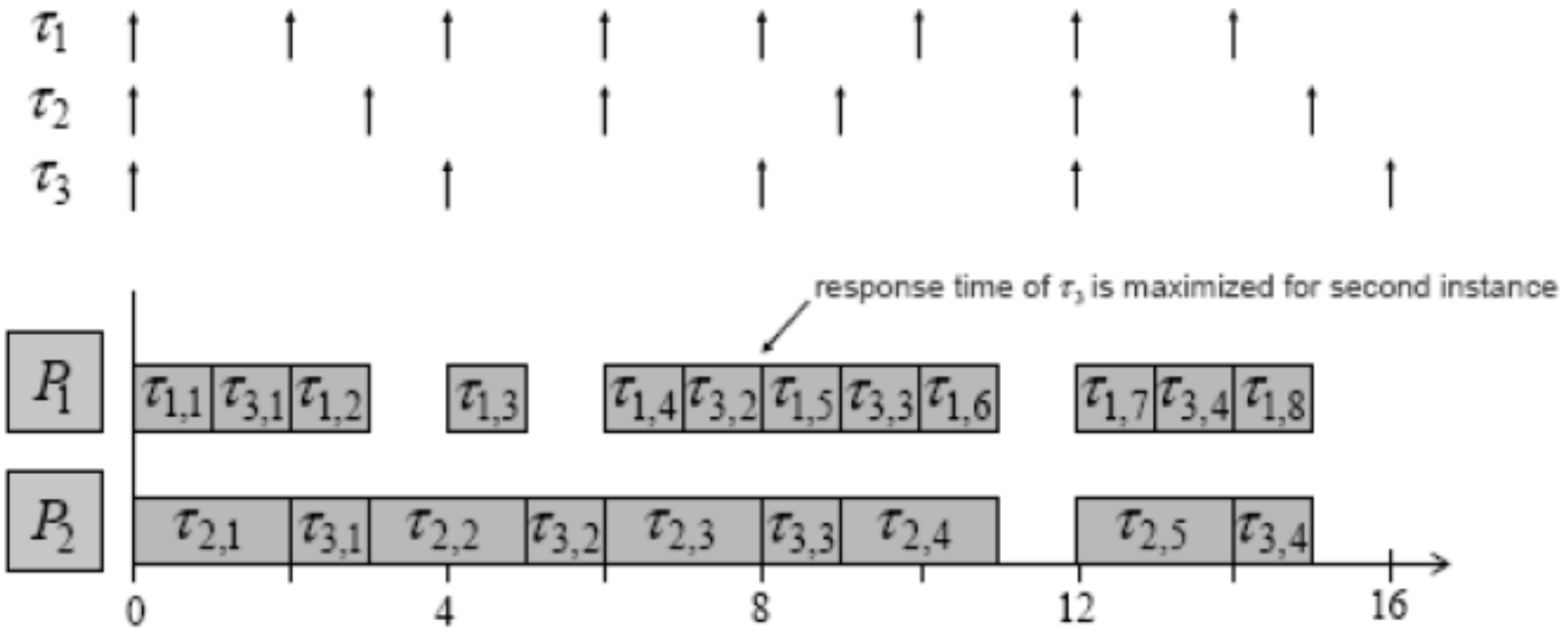
$$R_i = C_i + I_i$$

  - The difference with uniprocessor
    - Unknown critical instant
    - the calculation of interference now has to account for the fact that higher-priority tasks can execute in parallel with the analyzed task

Hard-to-Find Critical Instant

- with RM scheduling

$$\tau_1 = \{C_1 = 1, T_1 = 2\}$$
$$\tau_2 = \{C_2 = 2, T_2 = 3\}$$
$$\tau_3 = \{C_3 = 2, T_3 = 4\}$$



Hard-to-Find Critical Instant

- A critical instant does not always occurs when arrives at the same time as all its higher-priority
- Finding the critical instant is NP-complete
- Note: recall that knowledge about the critical instant is a fundamental property in uniprocessor feasibility tests

- Consequence: new methods for constructing effective multiprocessor feasibility tests are needed!

Poor Resource Utilization

- A fundamental limit:

  The ulitization guarantee bound for any static-priority multiprocessor scheduling algorithm can not be higher than ½ of the capacity of the processors.

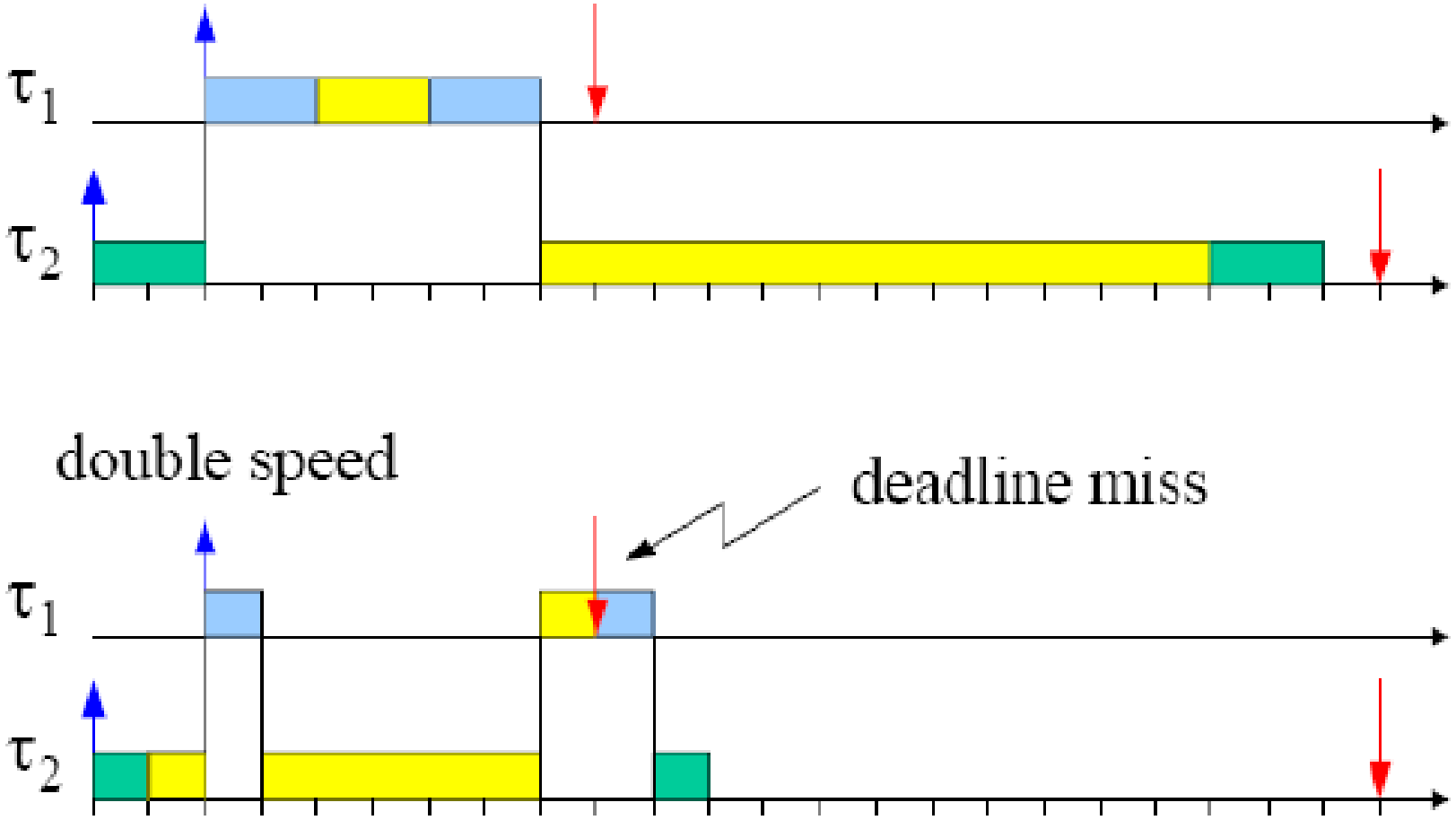
- This applies for all types of static-priority scheduling. (partitioned and global)
- Hence, we can never expect to always utilize more than half the processing capacity if hard timing constraints exist

MP Scheduling Anomalies

- Adding processors and reducing computation times and other parameters can actually decrease optimal performance with some scenarios!

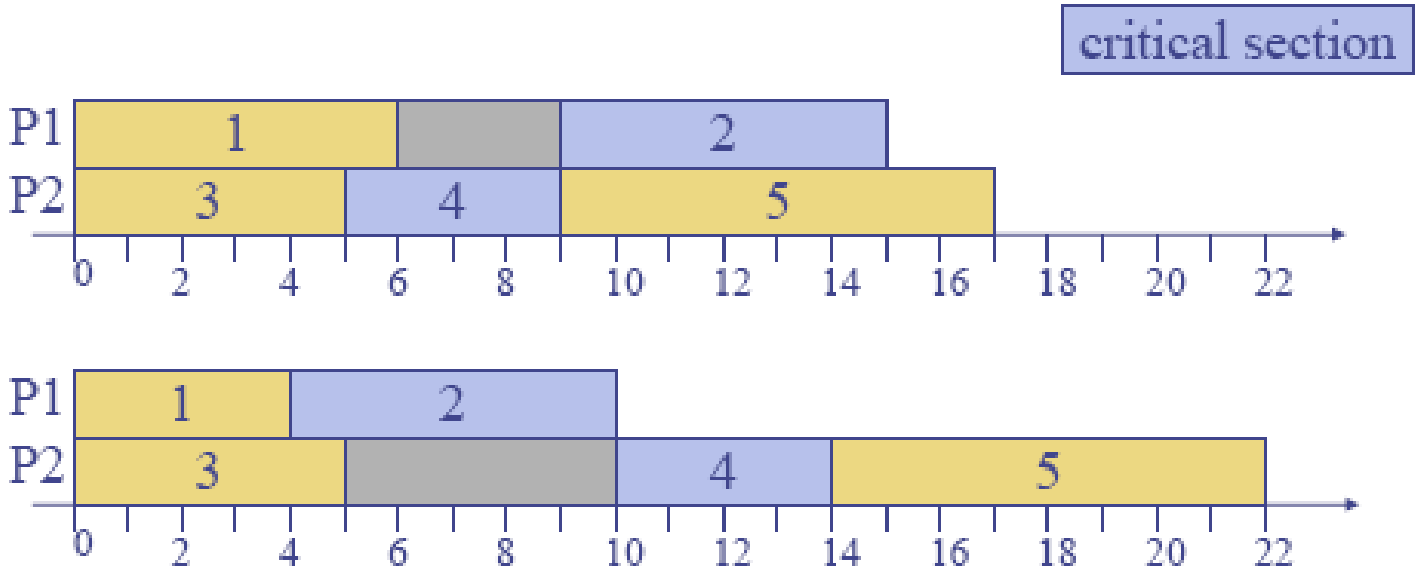
- EDF does not suffer from execution-time anomalies, but does suffer from period anomalies.

Doubling Processor Speed



Anomalies under Resource Constraints

- Task set of five tasks on two processors
- Task 2 and 4 share the a resource in exclusive mode
- Static allocation P1 (1,2) and P2 (3,4,5)
- Reducing the computation time of task 1 will increase the optimal schedule time!



This Chapter

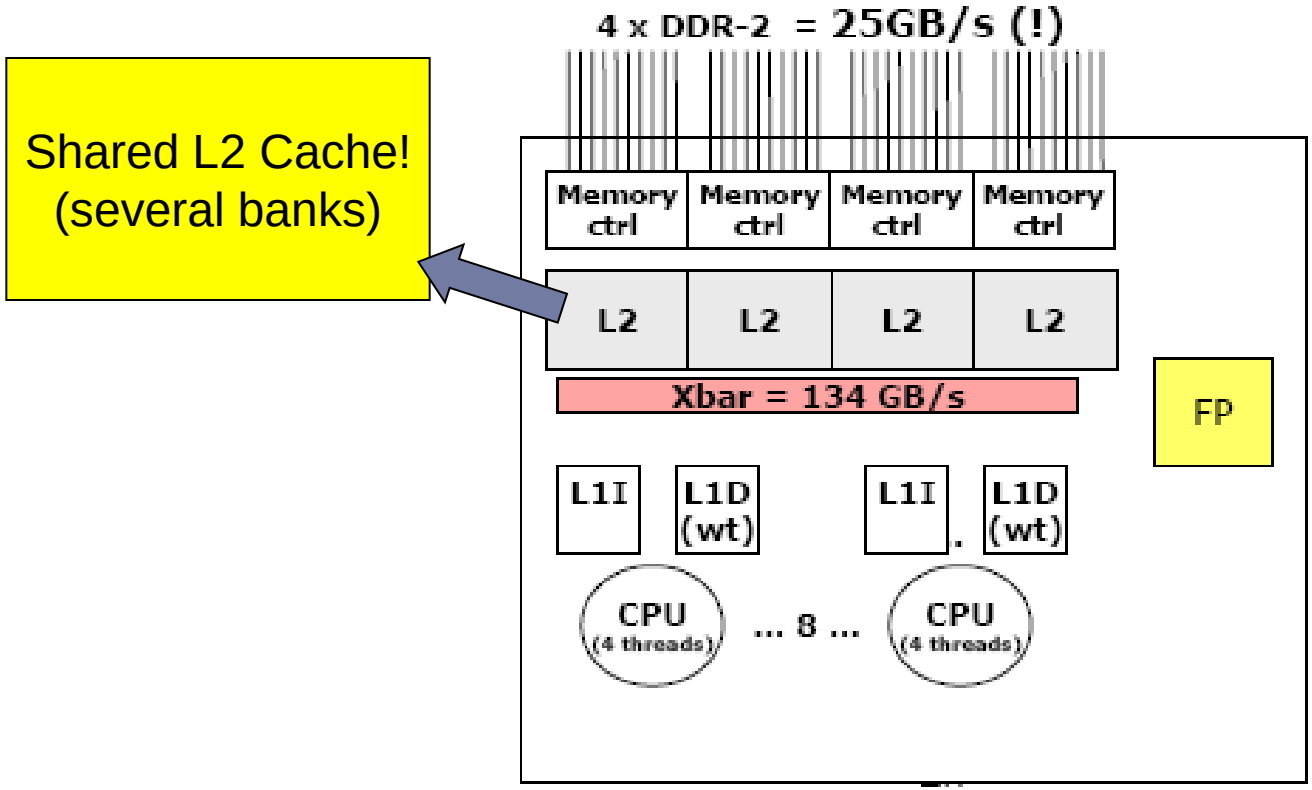
- Theoretic Aspect:
  Multiprocessor scheduling foundations
  (boring, but important to get insights!)

- Practical Aspect:
  Interesting issues of building real time systems on multi-core processors

Typical Multicore Architectures

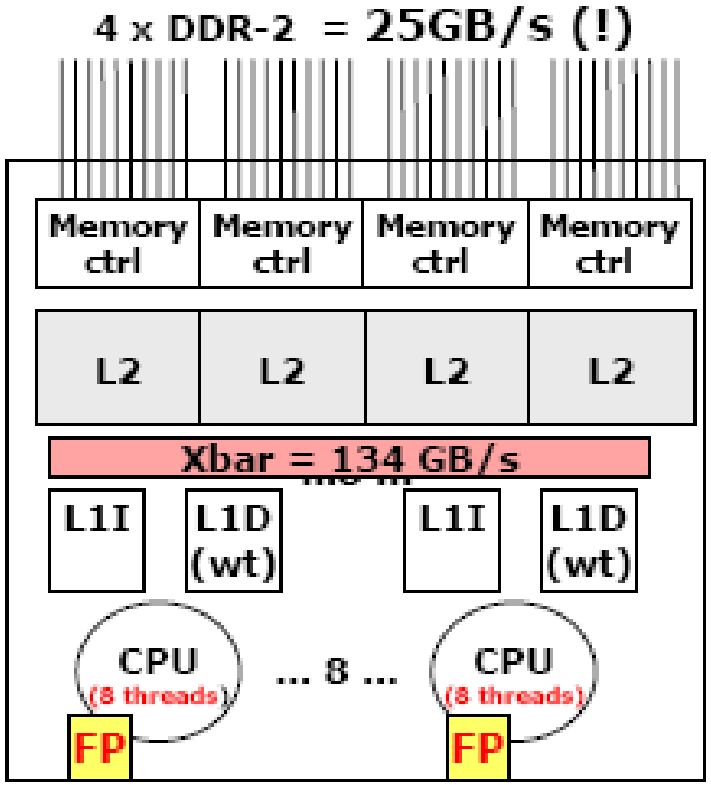
- **Niagara**

  (which is usually considered to be the start of modern multicore processors)



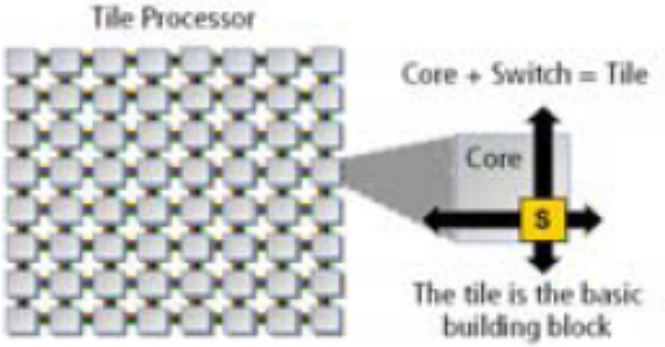
Typical Multicore Architectures

- **Niagara 2**



Typical Multicore Architectures

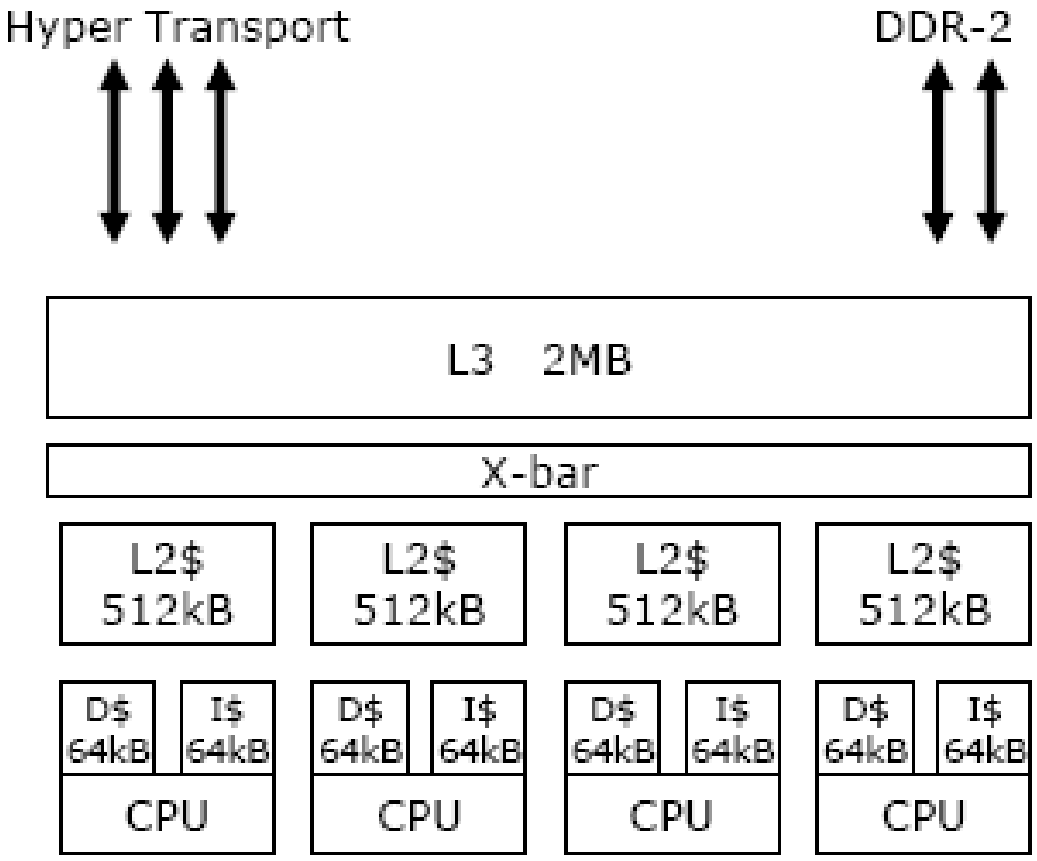
- **TILERA (Network-on-Chips)**



64 cores connected in a mesh
Local L1 + L2 caches
Shared distributed L3 cache
Linux + ANSI C
New Libraries
New IDE
Stream computing
...

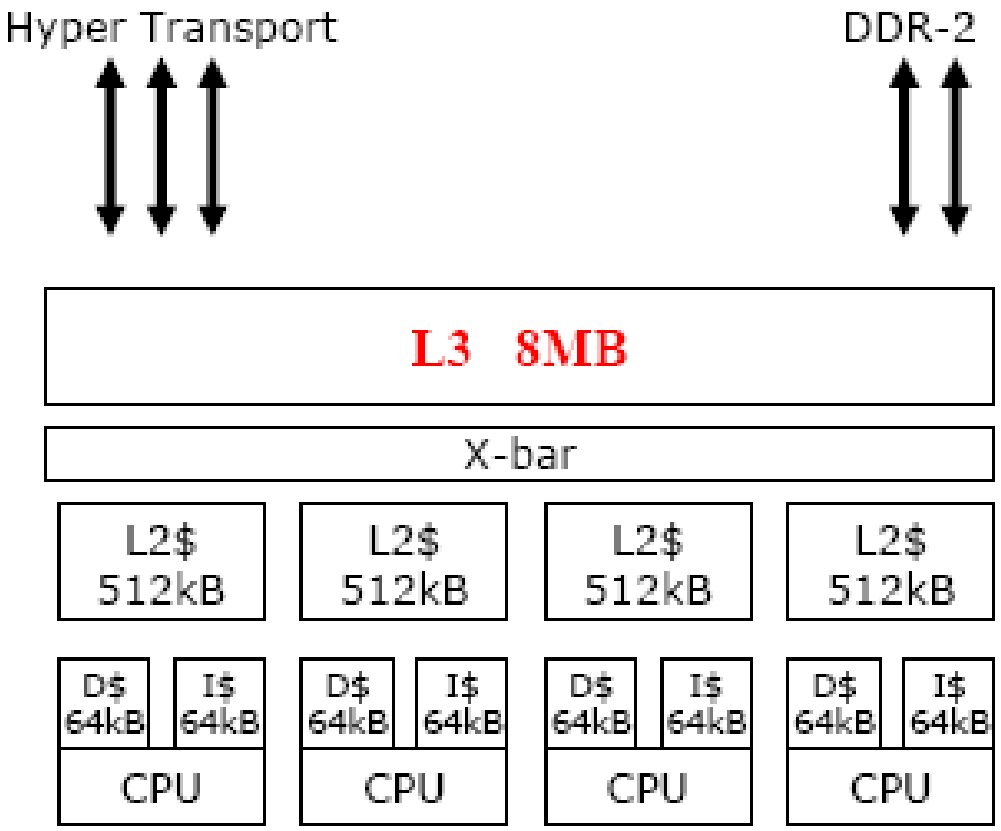
Typical Multicore Architectures

- **AMD Barcelona, 65 nm**



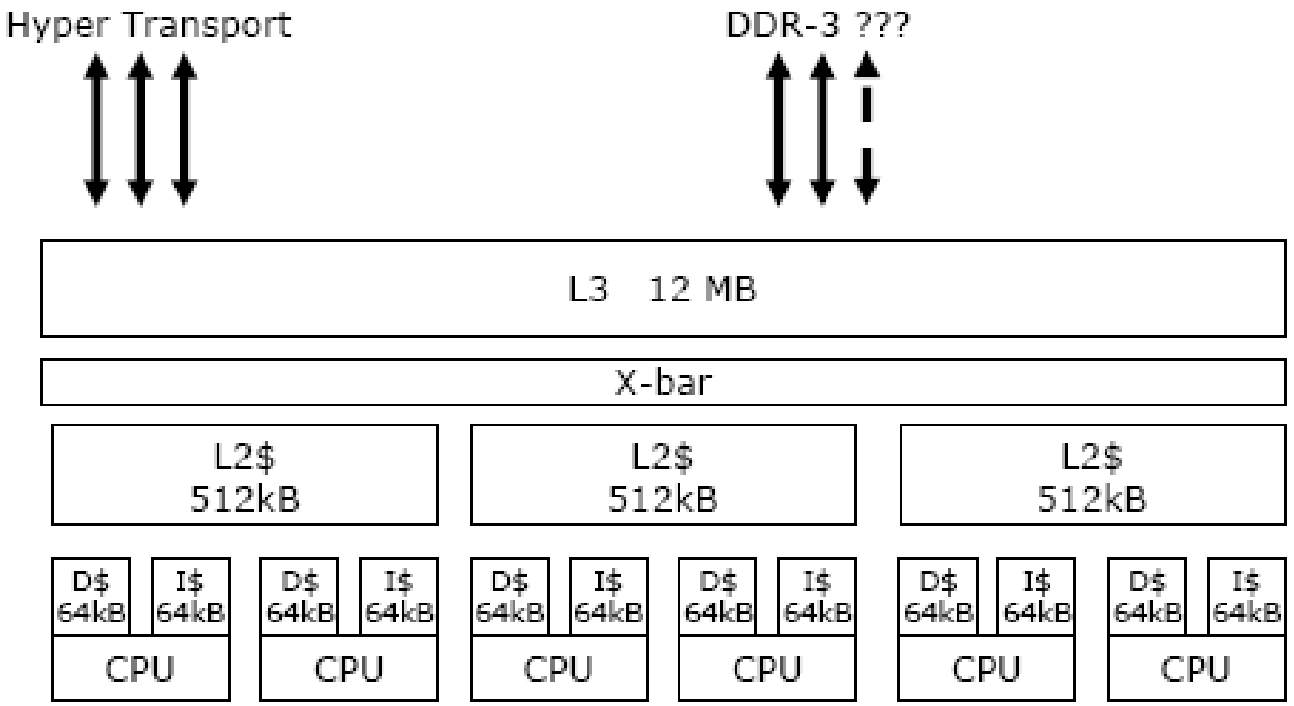
Typical Multicore Architectures

- **AMD Shanghai, 45 nm**



Typical Multicore Architectures

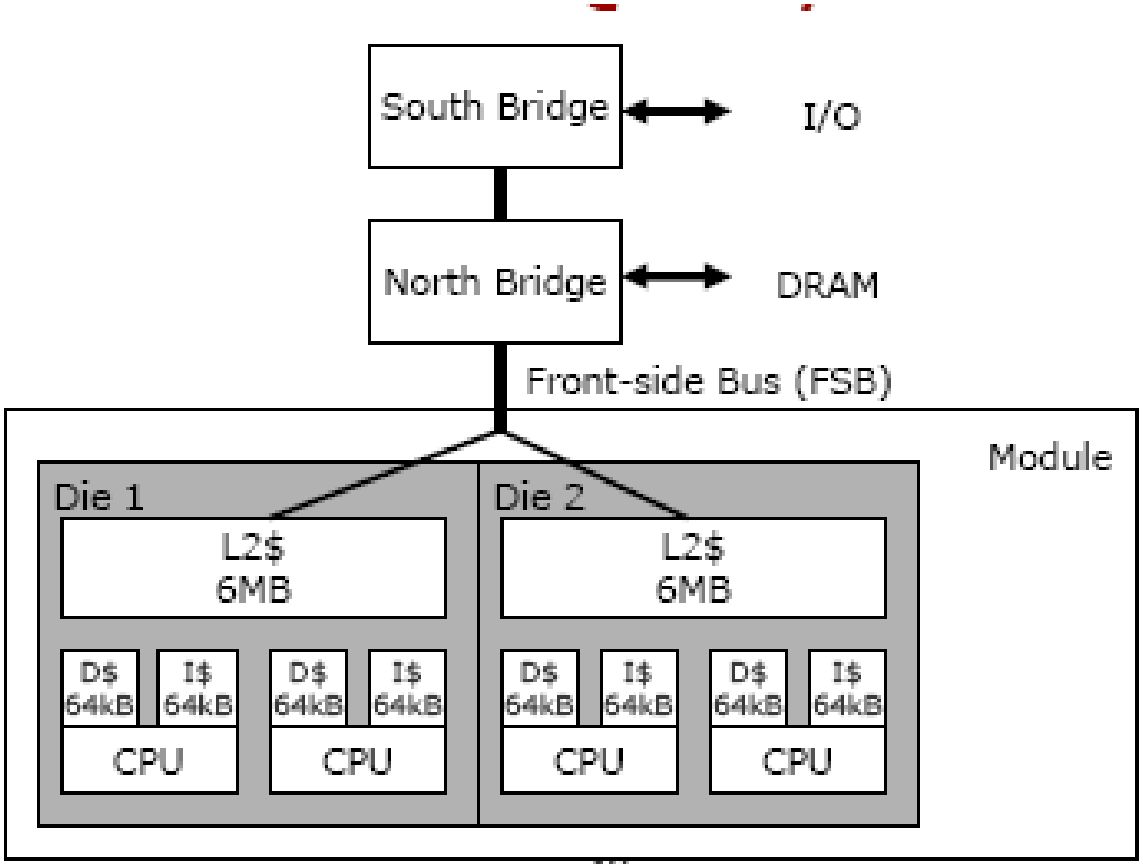
- **AMD Istanbul, 45 nm (coming soon)**



from the network, not based on the fact

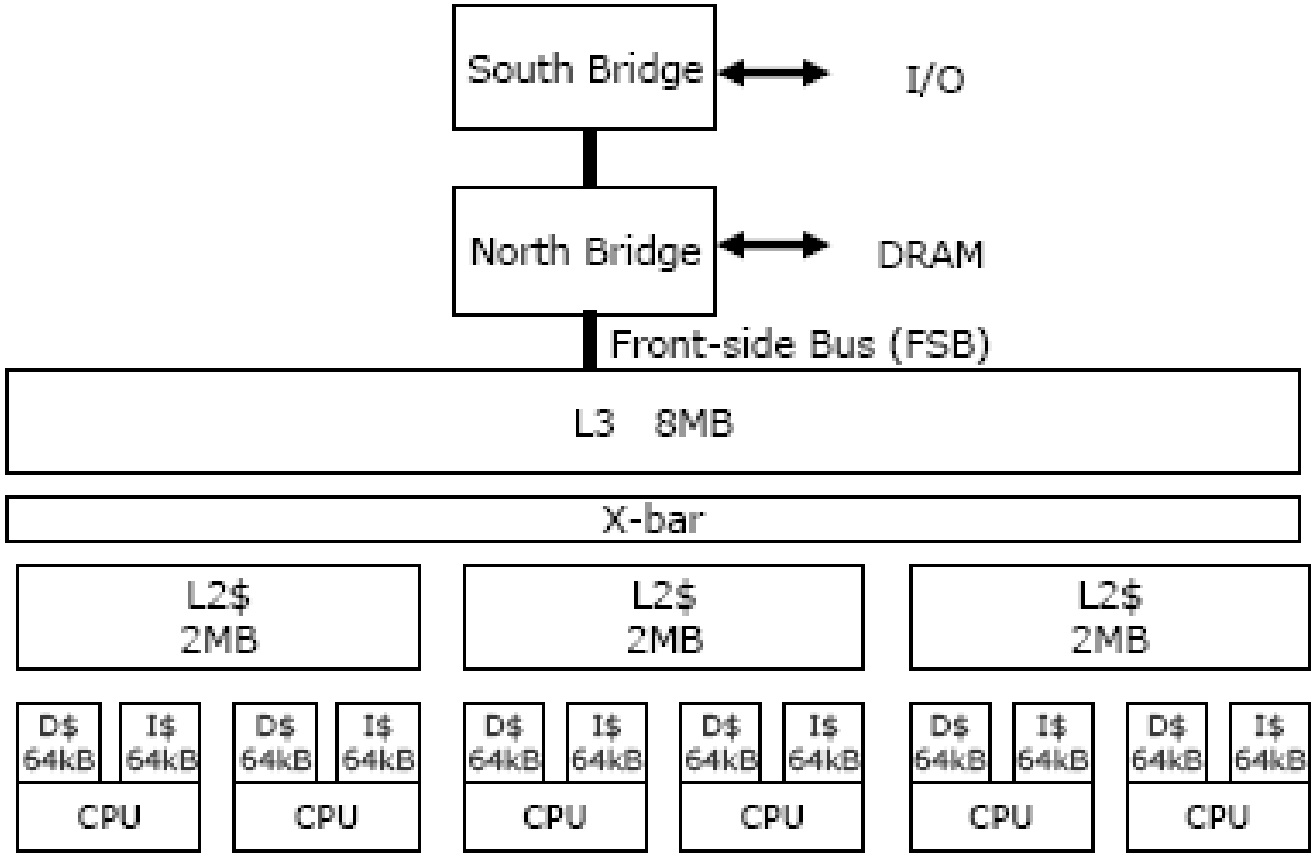
Typical Multicore Architectures

- **Intel Core2 Quad, 45 nm**



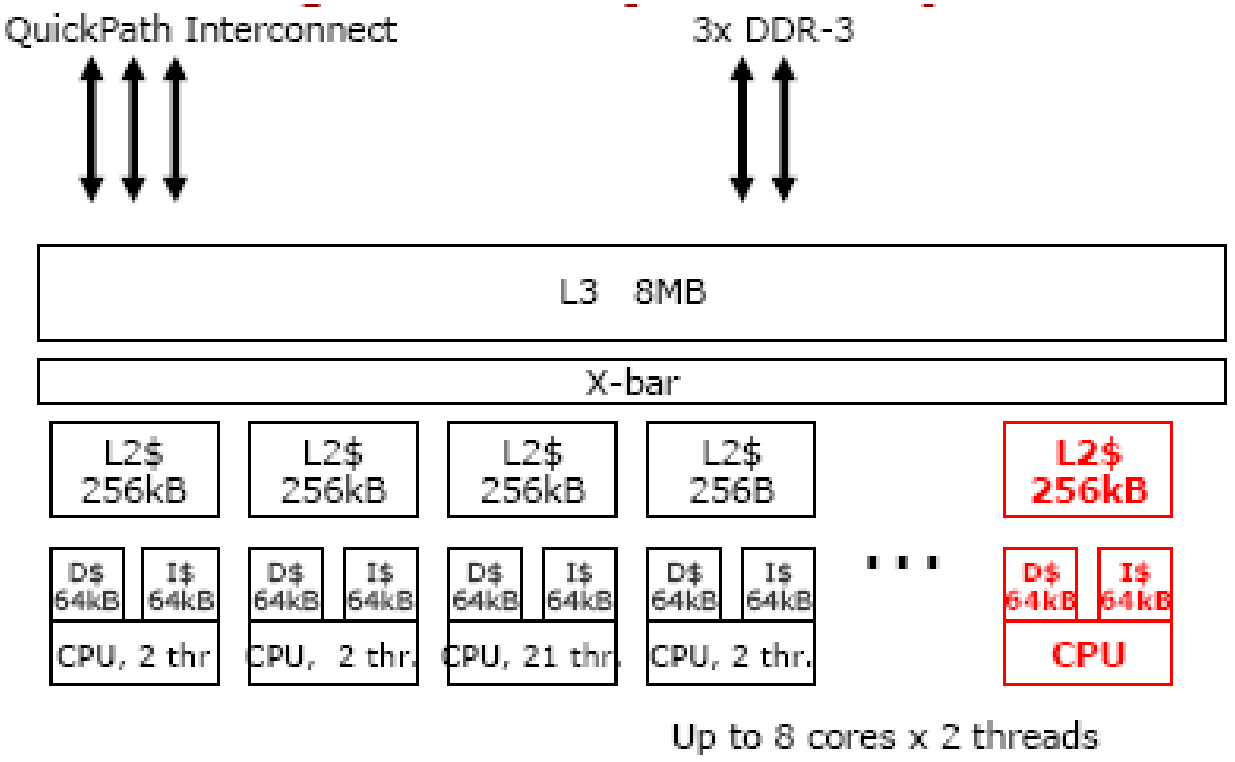
Typical Multicore Architectures

- **Intel: Dunnington, 45nm**



Typical Multicore Architectures

- **Intel: Nehalem, Core i7, 45 nm**



New Challenges

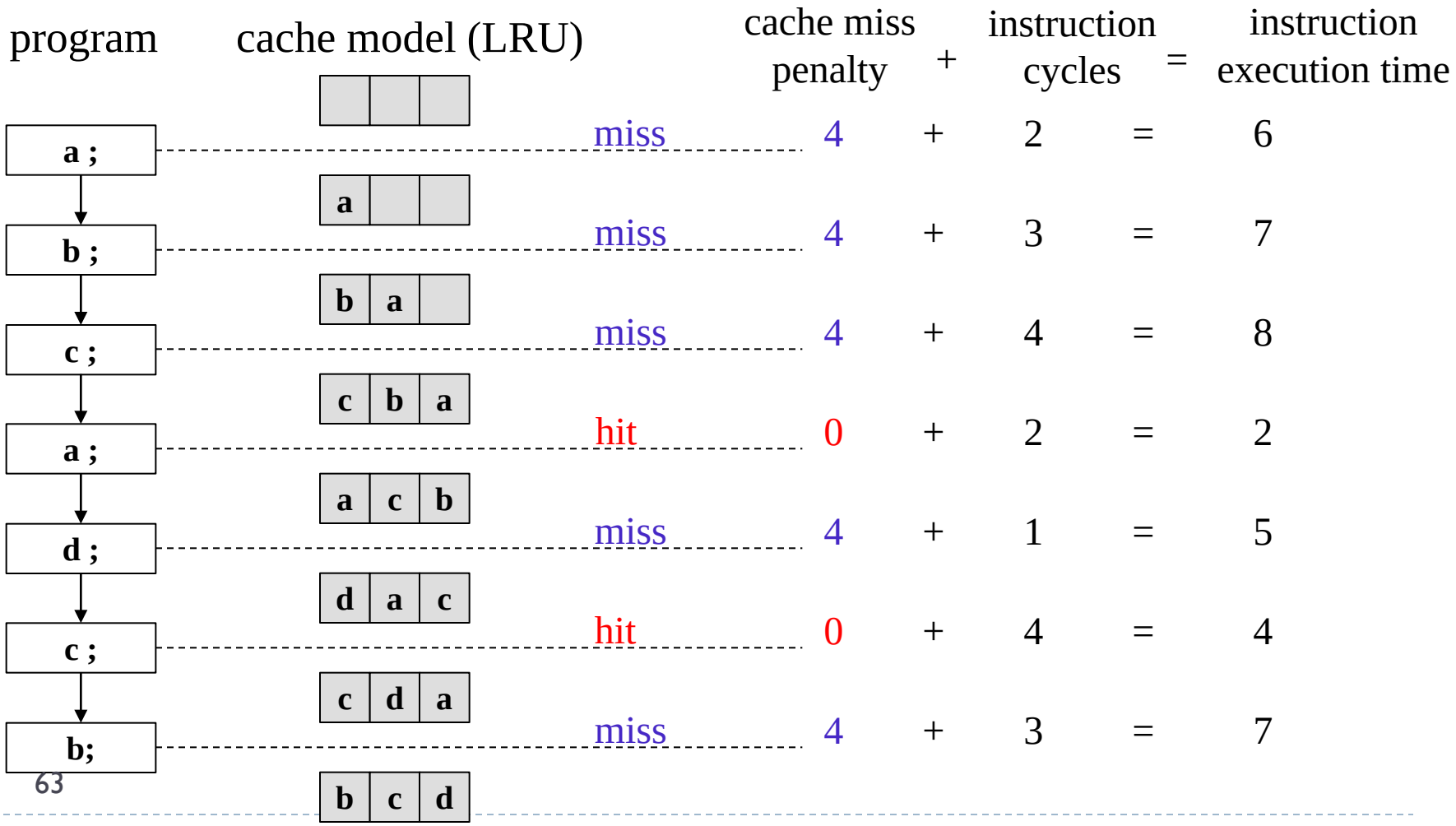
- Shared Computation Resources
  - Shared L2 Cache
  - Shared Bus
- Interference among Cores

Example

- Worst Case Execution Time (WCET)
  - the foundation of system-level timing analysis schedulability analysis, response time analysis, …
  - cache behavior modeling and analysis
    - cache hit/miss lead to different execution time for each instruction
    - well-studied in single-processor systems

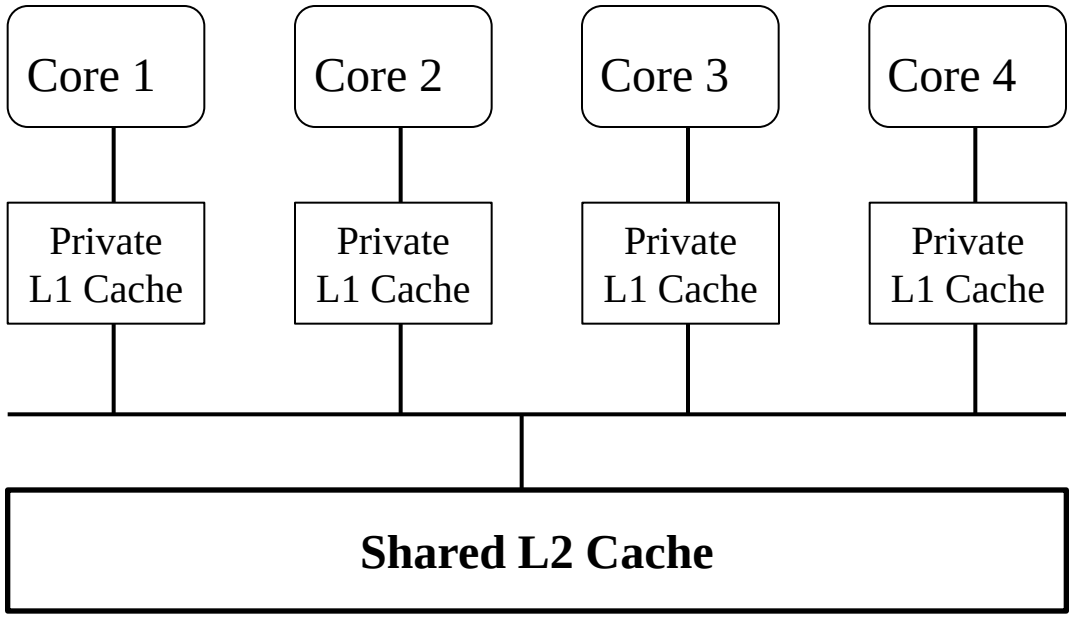
Example

- the brief idea of cache analysis



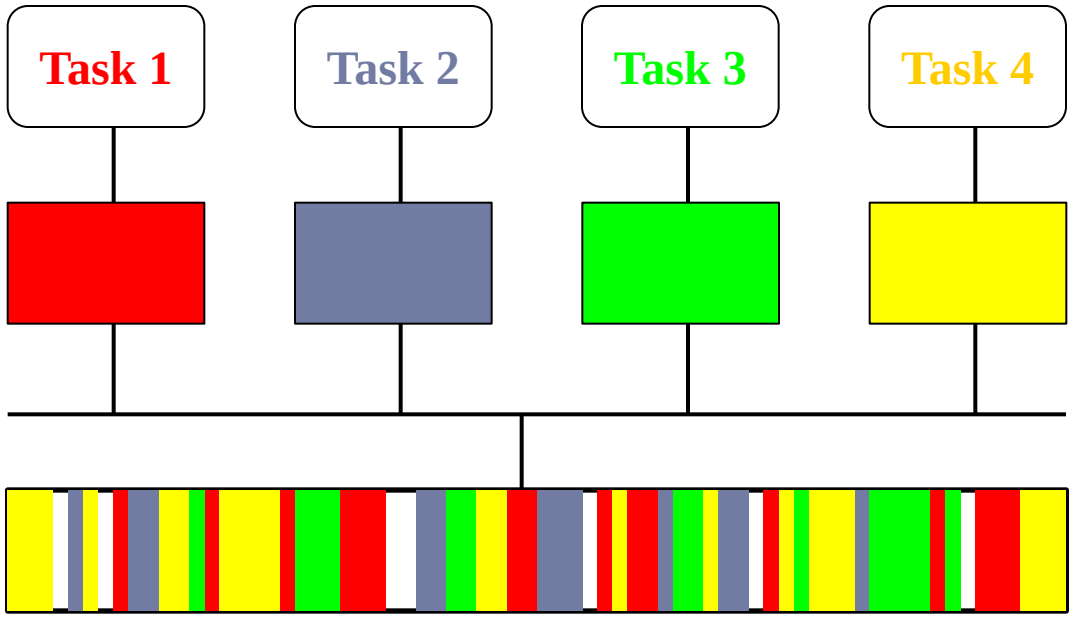
Example

- the typical multi-core architecture
  - shared L2 cache



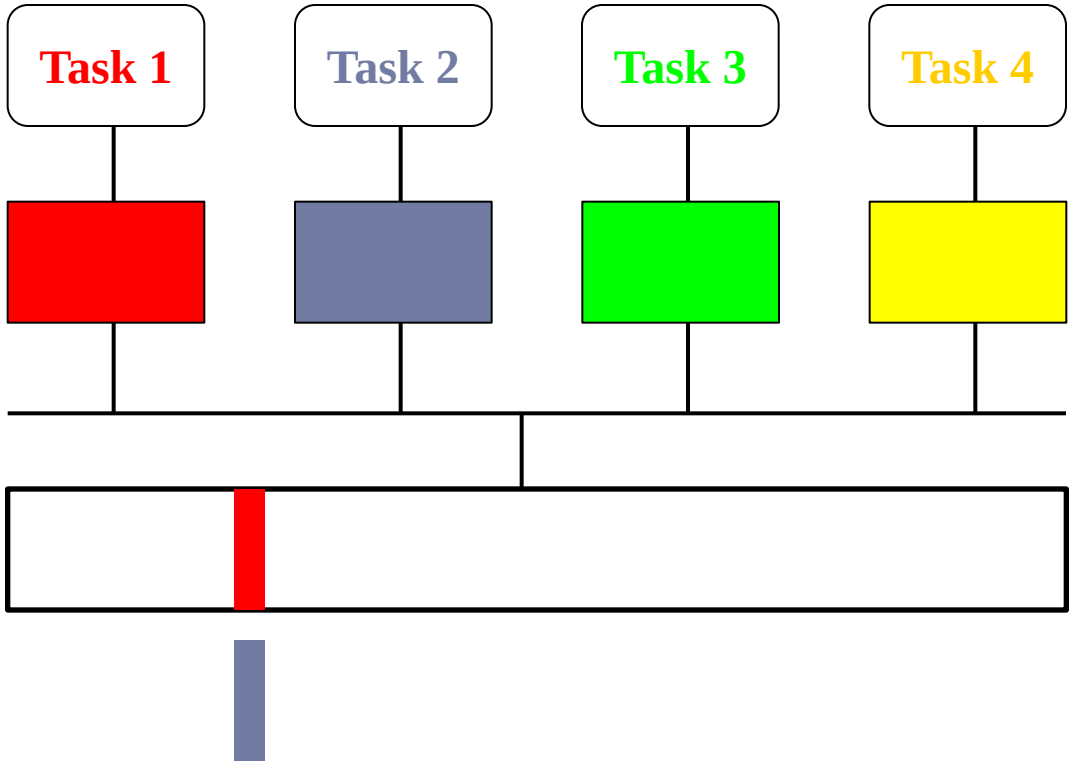
Example

- the typical multi-core architecture
  - shared L2 cache



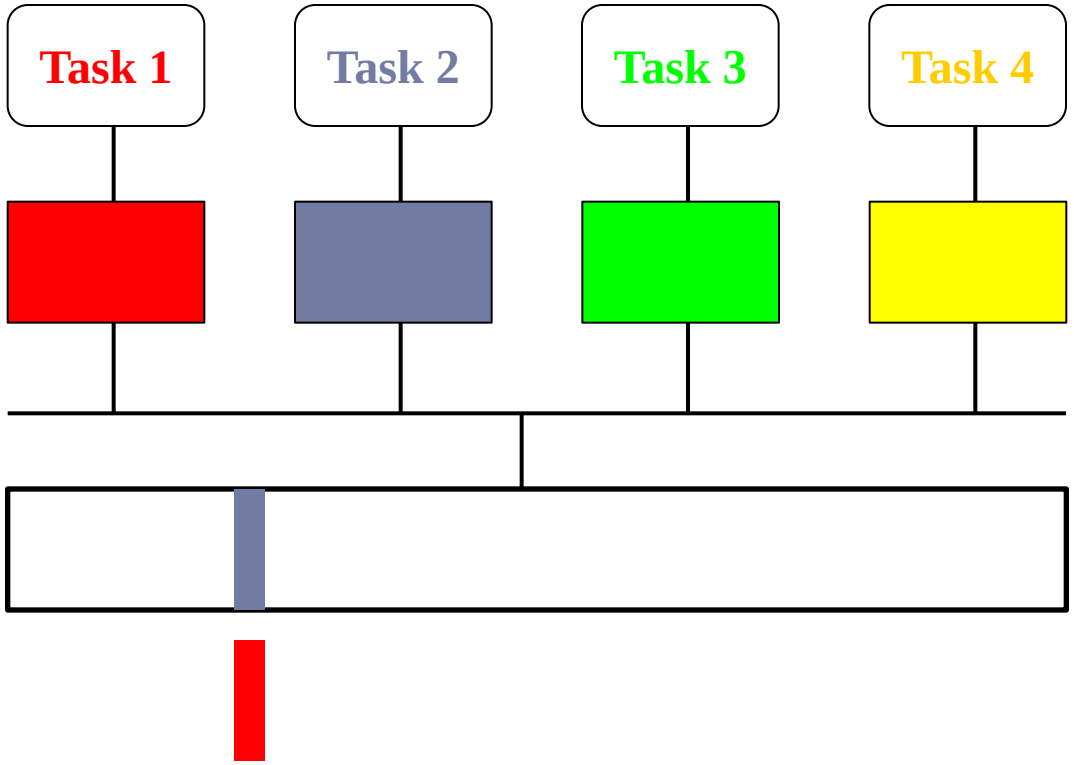
Example

- the typical multi-core architecture
  - the content belonging to task 1 may be evicted by the content belonging to task 2



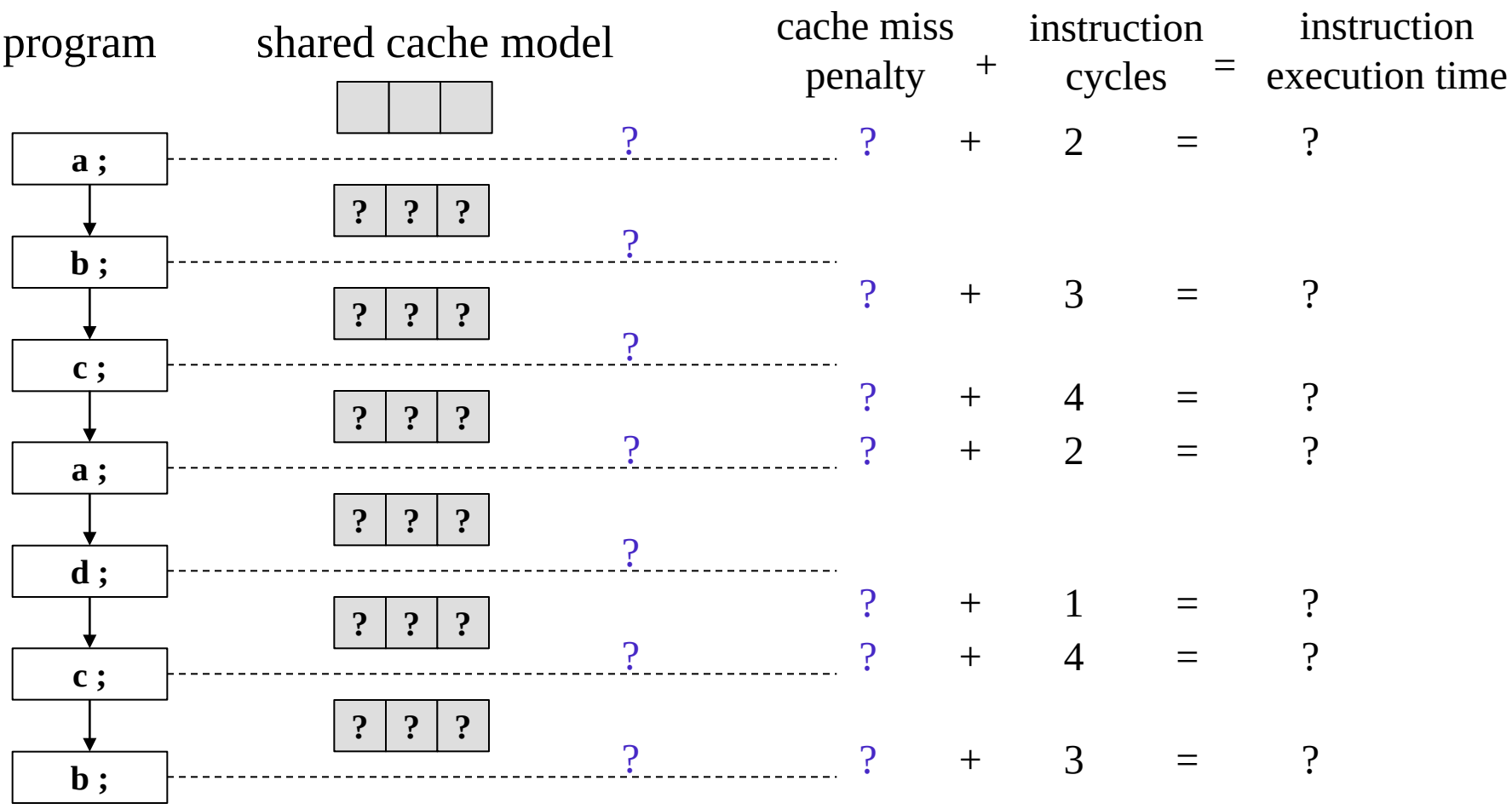
Example

- the typical multi-core architecture
  - the content belonging to task 1 may be evicted by the content belonging to task 2



Example

- for one task

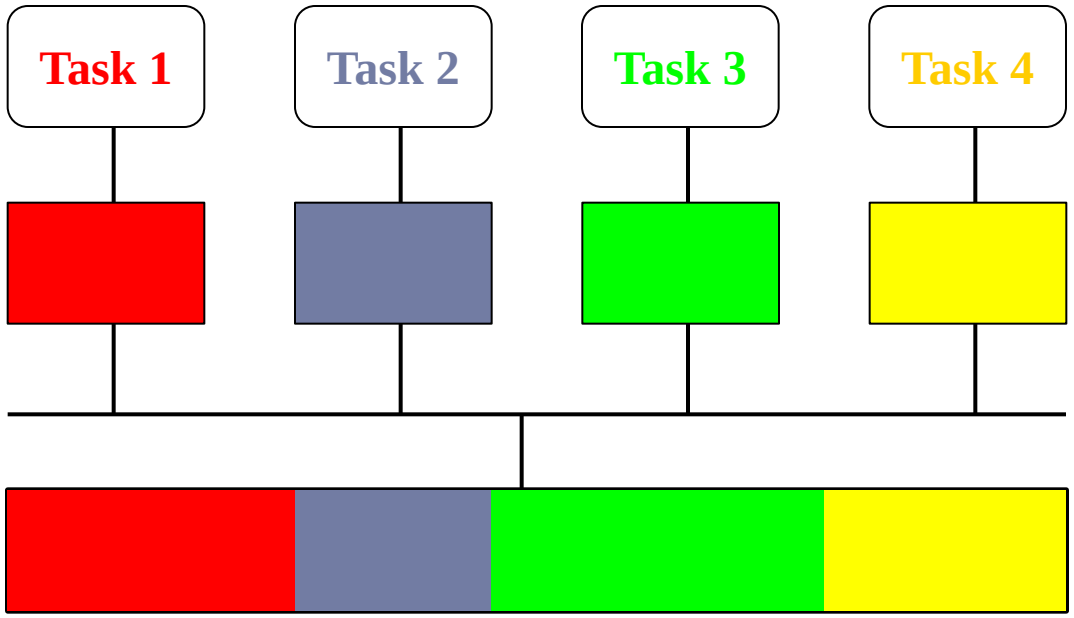


Example

- WCET analysis in presence of shared cache
  - Cache behavior is more unpredictable
    - Freely interleaving of instructions on different cores
  - Precise analysis is extremely difficult
    - Huge state space
    - Un-completed information

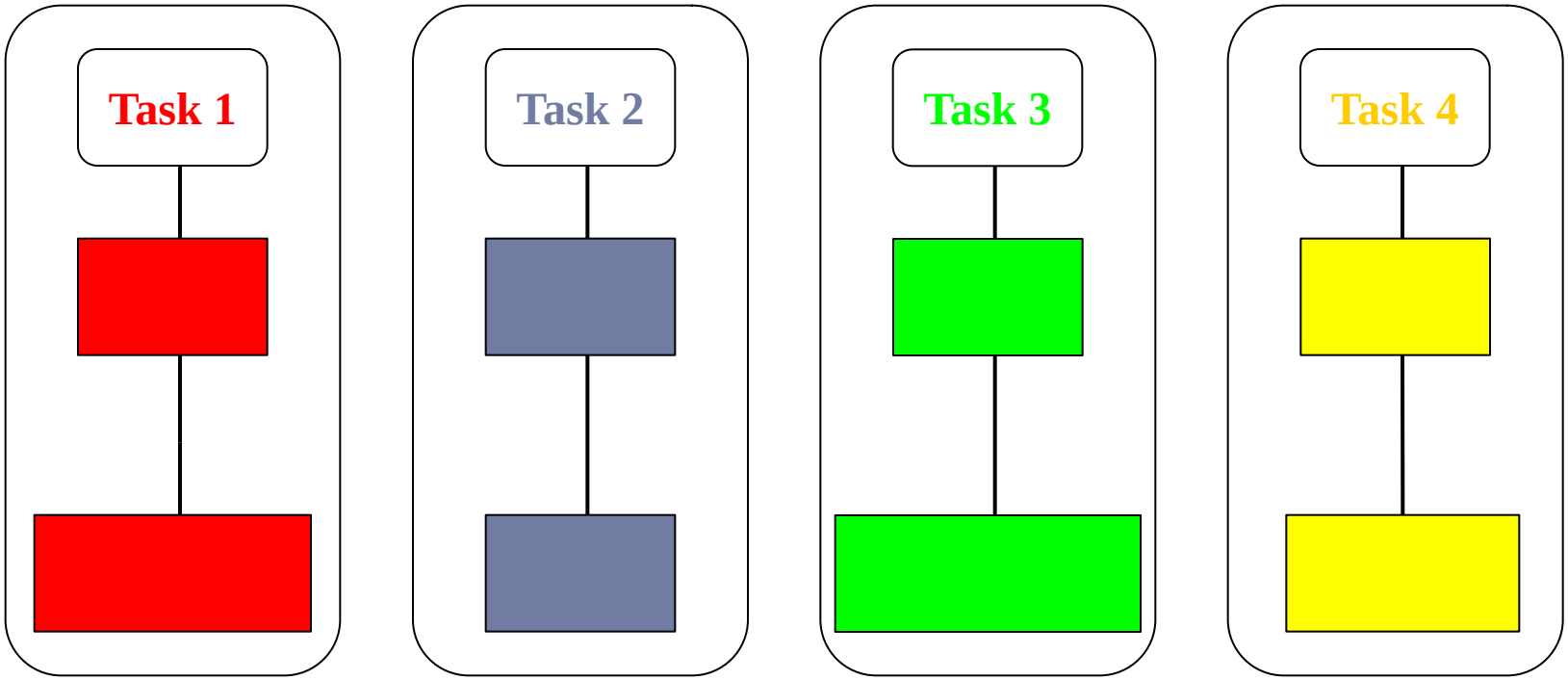
Solutions?

- cache space isolation to avoid interference between tasks



Solutions?

- therefore we can apply traditional WCET analysis techniques to each task

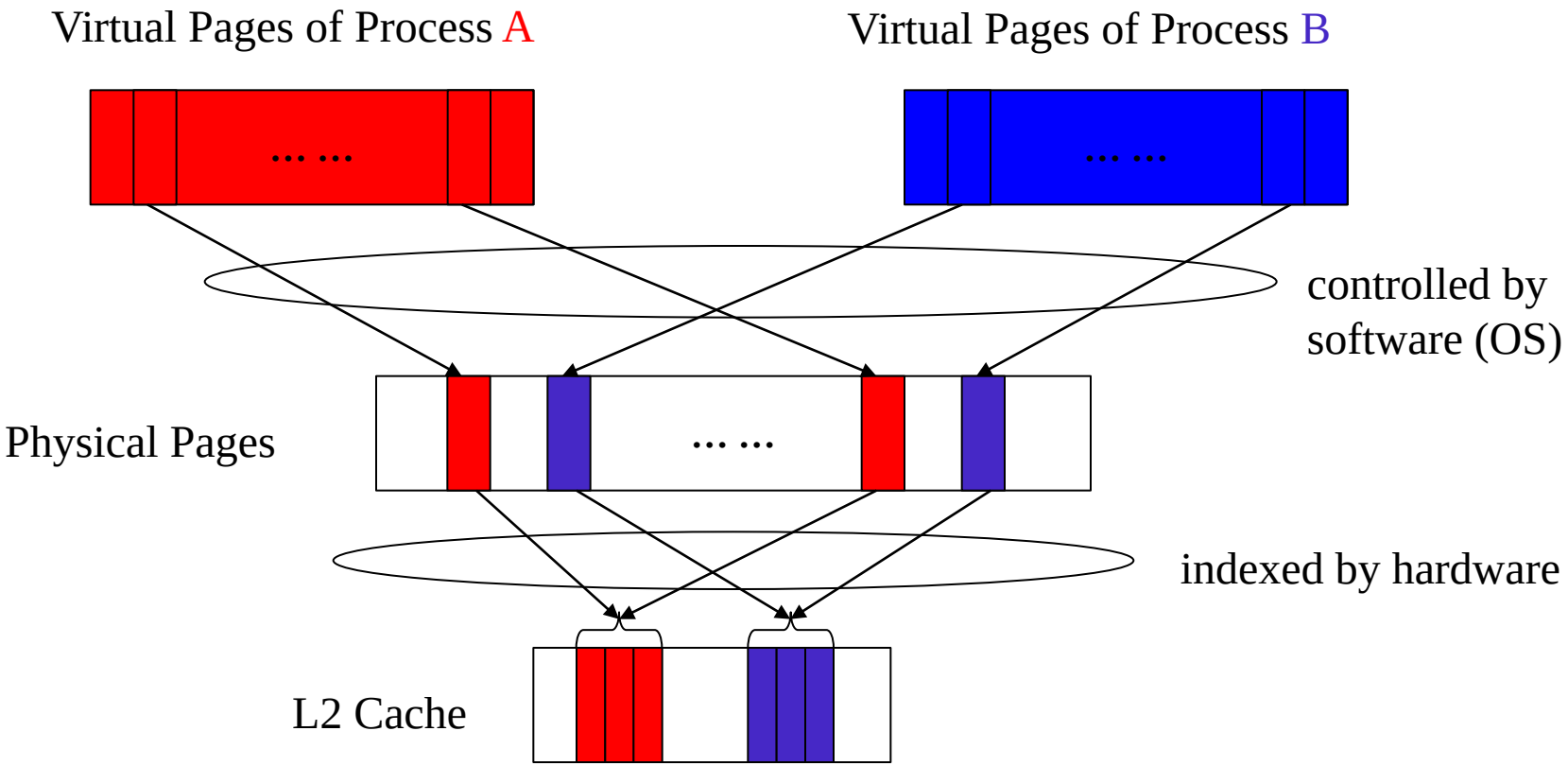


Cache Space Isolation

- How to implement Cache Space Isolation?
  - Hardware-based methods
  - Software-based methods
    - Page-coloring

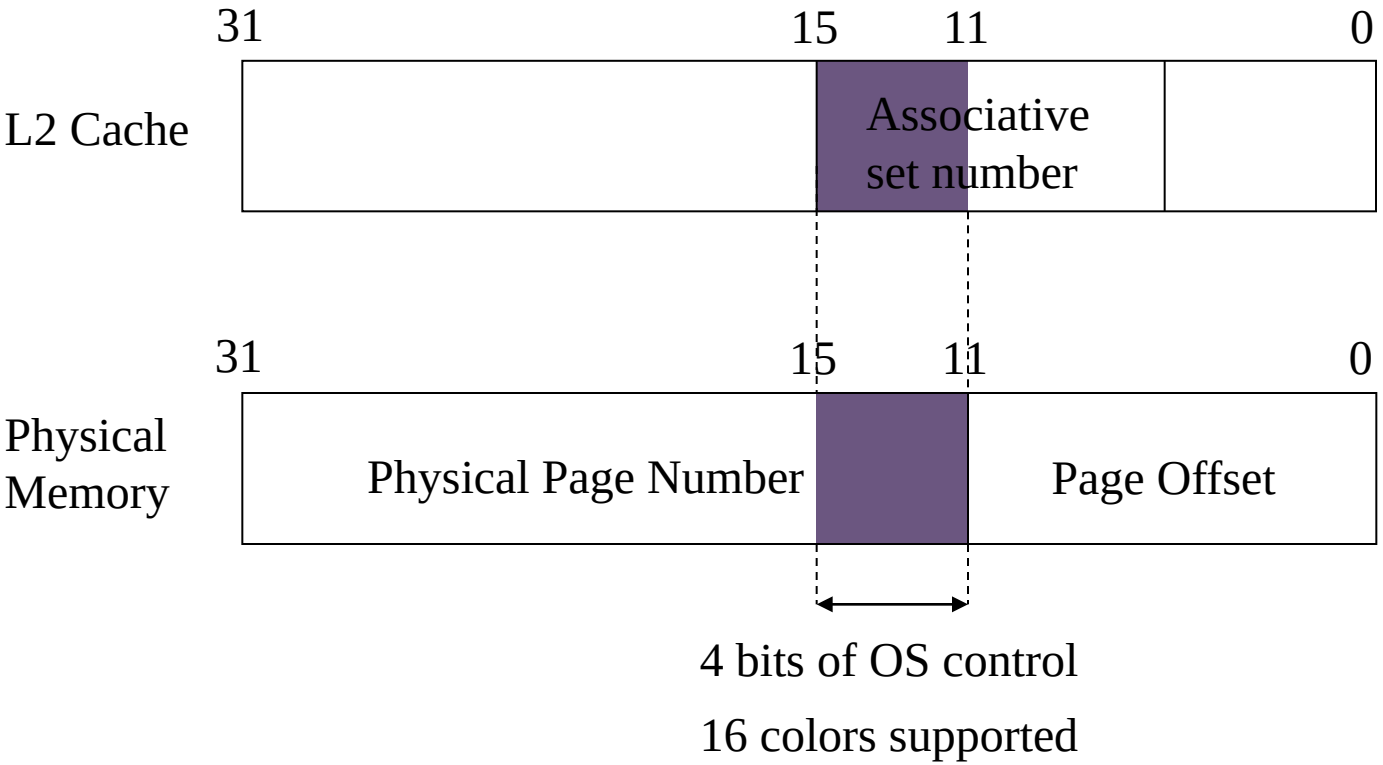
Cache Space Isolation

- Page coloring

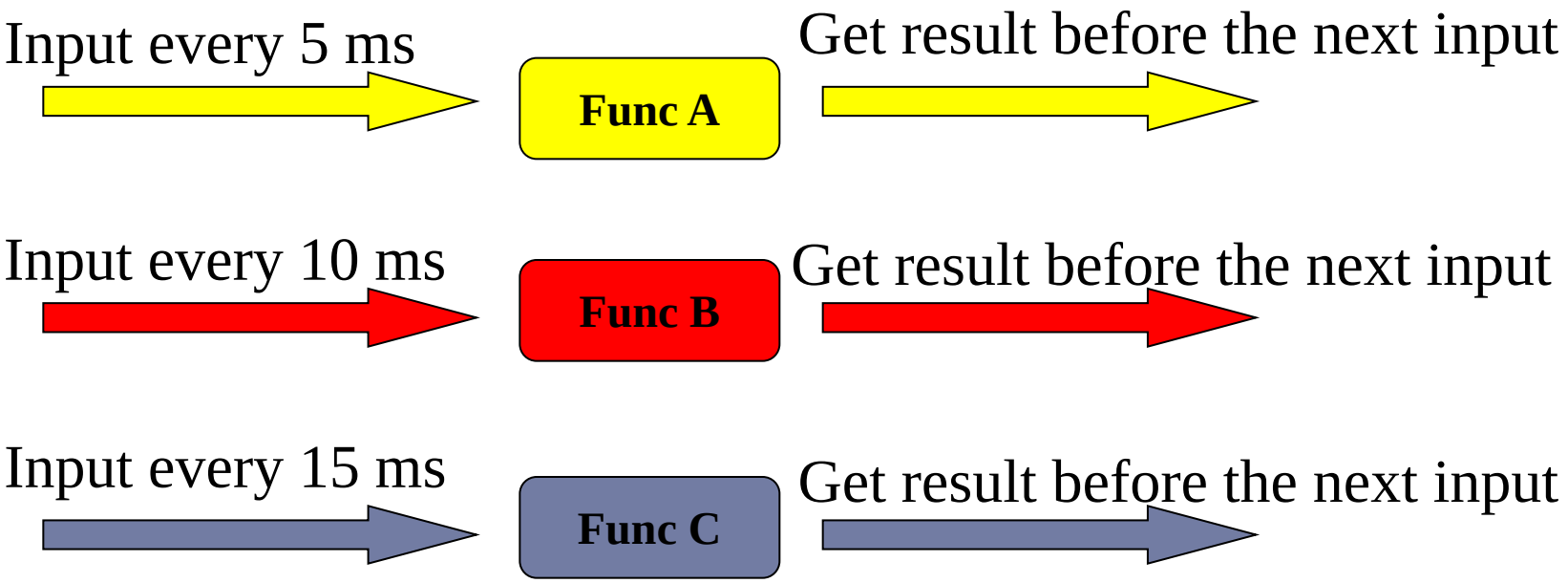


Cache Space Isolation

- Page coloring
  - address bit-view (Linux + Power 5)

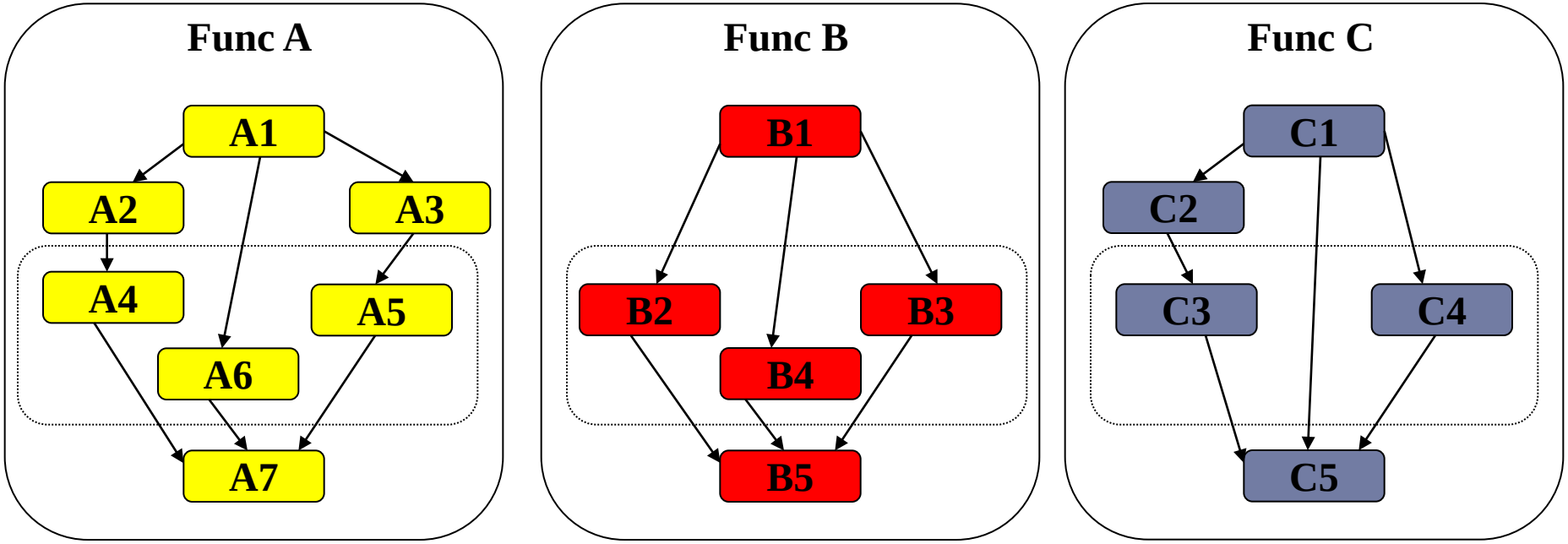


Co-runner


Input every 5 ms
Func A
Get result before the next input
Input every 10 ms
Func B
Get result before the next input
Input every 15 ms
Func C
Get result before the next input


intel
Core 2
Duo
inside

Co-runner



Co-runner

- the cache (memory) requirement of executing

  is higher than running

- the concept of

  "good" co-runner

  "bad" co-runner